HP-85

```
17
18
190
200
210 P
    ,"M
220 MOVE
230 FOR R
240 D=P^(N-
    *FNF(N-K
250 PRINT R,I      *1000+.5)/1000
260 DRAW R,D
270 IDRAW 1,0
280 NEXT R
290 DRAW 11,0
300 DEF FNF(X)
310 F=1
320 FOR I=X TO 1 STEP -1
330 F=F*I
340 NEXT I
350 FNF=F
360 FN END
```

NO % BIO 101 FINAL EXAM GRADES
20 10
18 9
16 8
14 7
12 6
10 5
8 4
6 3
4 2
2 1
0 0
E
D
C
B
A
LIM:
40
48
56
64
72
80
88
96

[redacted] [redacted]

[redacted]

HEWLETT-PACKARD

HP-85

OWNER'S MANUAL AND PROGRAMMING GUIDE

NET WEIGHT DISTRIBUTION
HP-85
HEWLETT PACKARD 85
NO % BIO 101 FINAL EXAM GRADES
20 18 16 14 12 10 8 6 4 2 0
10 9 8 7 6 5 4 3 2 1 0
E D C B A
LIM: 40 48 56 64 72 80 88 96

```
260 DRAW R,D
270 IDRAW 1,0
280 NEXT R
290 DRAW 11,0
300 DEF FNF(X)
310 F=1
320 FOR I=X TO 1 STEP -1
330 F=F*I
340 NEXT I
350 FNF=F
360 FN END
```

"The success and prosperity of our company will be assured only if we offer our customers superior products that fill real needs and provide lasting value, and that are supported by a wide variety of useful services, both before and after sale."

Statement of Corporate Objectives
Hewlett-Packard

Hewlett-Packard Company
Corvallis Division
1000 N.E. Circle Boulevard
Corvallis, Oregon 97330
(For world-wide sales and service offices see back of manual)

HEWLETT
PACKARD

HP-85
Owner's Manual
and
Programming Guide

July 1980

00085-90002 Rev. C 7/80

Printed in U.S.A.

©Hewlett-Packard Company 1979

Contents

Meet the HP-85 Personal Computing System 7
- How to Use This Handbook 7
- An Overview of the Hewlett-Packard 85 10
- HP-85 Key Index 12

Part I: Using Your HP-85 15

Section 1: Getting Started 17
- Power On 17
- Manual Problem Solving ("Calculator" Mode) 18
  - Simple Display Editing 19
  - Clearing the Display 19
  - Error Messages and Warnings 20
  - Variables 21
- Running a Prerecorded Program 22
  - Loading a Program From the Standard Pac 22
  - Halting Program Execution 26
- Writing Your Own Programs 26
  - Creating the Program 27
  - Entering the Program 27
  - Running the Program 27
  - An Averaging Program 28
  - Recording the Program 30
- Erasing a Program From the Tape Cartridge 31

Section 2: Keyboard, Printer, and Display Control 33
- The Keyboard 33
  - Typewriter Keys 33
  - BASIC Typewriter Mode 33
  - Normal Typewriting Mode 34
- HP-85 Character Set 34
- Printer Control 35
- The Display 36
  - Entering Long Expressions 36
  - Display Editing 38
    - Fast Backspace 38
    - Deleting Characters 38
    - Inserting Characters 39
- System Self-Test 39
- Resetting the Computer 40

Section 3: Expressions and Keyboard Operations 43
- Keyboard Arithmetic 43
  - MOD and DIV 44
  - Arithmetic Hierarchy 45
  - Parentheses 45
  - The RESULT Key 46
- PRINT and DISP 46
- Standard Number Format 47
  - Scientific Notation 48
  - Keying In Exponents of Ten 48
  - Range of Numbers 49
- Variables 49
  - Types 49
  - Forms 49
  - Simple Variables 50
  - String Variables 51
    - String Concatenation 52
    - The Null String 52
- Logical Evaluation 53
  - Relational Operators 53
  - Logical Operators 54
- The Time Functions 56

Section 4: Math Functions and Statements 59
- Number Alteration 59
  - Absolute Values 60
  - Integer Part of a Number 60
  - Fractional Part of a Number 60
  - Greatest Integer Function 60
  - Smallest Integer Function 61

General Math Functions 61
  Square Root Function 62
  Sign of a Number 62
  Maximum and Minimum 62
  The Remainder Function 62
  Using PI 63
  Epsilon and Infinity 64
  Random Numbers 64
Logarithmic Functions 65
Trigonometric Functions and Statements 66
  Trigonometric Modes 66
  Trigonometric Functions 66
  Degrees/Radians Conversions 67
  Polar/Rectangular Coordinate Conversions 67
Total Math Hierarchy 69
Recovering From Math Errors 69

Part II: BASIC Programming With Your HP-85 71

Section 5: Simple Programming 73
Loading a Prerecorded Program 74
  Stopping a Running Program 75
  Listing a Program 75
What Is a BASIC Program? 76
  Statements 76
  Statement Numbers 77
  Commands 77
Clearing Computer Memory 78
Writing a Program 78
Entering a Program 79
  Automatic Numbering 80
  Spacing 80
  Statement Length 81
  Entering Program Statements into Computer Memory 81
  Entering the Program 81
Running a Program 82
Order of Program Execution 83
Fundamental BASIC Statements 83
  REMarks 83
  DISPlay 84
  PRINT 86
  INPUT: Assigning Values From the Keyboard 87
  BEEP 89
  LET: Assignments 90
  GOTO: Unconditional Branching 91
Multistatement Lines 92
Problems 93

Section 6: Program Editing 95
Editing Program Statements 95
Deleting Statements 95
Adding Statements 96
Renumbering a Program 96
Listing a Modified Program 96
Interrupting Program Execution 97
  Pausing 97
  Continuing 98
Initializing a Program 99
Using PAUSE in a Program 99
Delaying Program Execution 100
Error Messages 100
Problems 101

Section 7: Branches and Loops 103
Conditional Branching 103
The ELSE Option 106
The Computed GOTO Statement 108
FOR-NEXT Loops 110
  Changing the Increment Value 114
  Nested Loops 115
  FOR-NEXT Loop Considerations 116
Problems 116

Section 8: Using Variables: Arrays and Strings 119
Array Concepts 119

Declaring and Dimensioning Variables ... 121
  Lower Bounds of Arrays ... 121
  The DIM Statement ... 121
  Type Declaration Statements ... 122
  The COM Statement ... 123
  About Variable Declarations ... 123
String Expressions ... 124
  Substrings ... 125
  Modifying String Variables ... 125
    Replacing a String ... 126
    Replacing Part of a String ... 126
  String Functions ... 128
    The Length Function ... 128
    The Position Function ... 129
    Converting Strings to Numbers ... 130
    Converting Numbers to Strings ... 131
    Character Conversions ... 131
      Numbers to Characters ... 131
      Characters to Numbers ... 132
      Lowercase to Uppercase Conversion ... 133
Assigning Values to Variables in a Program ... 133
  Assigning Values to Array Elements ... 133
  Initializing Variables ... 136
  The READ and DATA Statements ... 137
  Rereading Data: The RESTORE Statement ... 139
System Memory and Variable Storage ... 140
  Memory ... 140
  Storing Variables ... 141
  Conserving Memory ... 141
Problems ... 142

Section 9: More Branching ... 145
Defining a Function ... 145
  Single-Line Functions ... 145
  Multiple-Line Functions ... 147
Subroutines ... 151
The Computed GOSUB Statement ... 153
Branching Using Special Function Keys ... 154
  KEY LABEL ... 154
  Cancelling Key Assignments ... 156
The Timers ... 156
Problems ... 158

Section 10: Printer and Display Formatting ... 161
Using IMAGE ... 161
Delimiters ... 161
Blank Spaces ... 162
String Specification ... 162
Numeric Specification ... 163
  Digit Symbols ... 163
  Radix Symbols ... 164
  Sign Symbols ... 164
  Digit Separator Symbols ... 165
  Exponent Symbol ... 165
Compacted Field Specifier ... 166
Replication ... 166
Reusing the IMAGE Format String ... 166
Field Overflow ... 167
Formatting in PRINT/DISP USING Statements ... 167
The TAB Function ... 168
Redefining the Printer and the Display ... 169
Problems ... 170

Section 11: Using Tape Cartridges ... 175
The Tape Directory ... 175
  Directory Set-up ... 175
  Cataloging ... 175
Recording and Retrieving Programs ... 176
  The STORE Command ... 176
  The LOAD Command ... 179
  The CHAIN Statement ... 179
Autostart ... 180
Using Data Files ... 180
Creating a Data File ... 180
  Records ... 180

    Data Storage 181
  Opening a Data File 183
  Closing a Data File 184
  Storing and Retrieving Data 185
    Serial File Access 185
      Writing Serial Files 185
      Reading Serial Files 187
    Random File Access 188
      Random Writing 188
      Random Reading 190
      Repositioning the Pointer 190
  Storing and Retrieving Arrays 191
  Purging a File 192
  Renaming a File 192
  Binary Programs 193
  Securing Files 193

Section 12: Graphics 197
  The Graphics Display 197
  Line Generation 198
  Graphics and the Printer 198
  Clearing the Graphics Display 199
  Setting Up the Graphics Display 199
    The SCALE Statement 199
      Unequal Unit Scaling 200
      Equal Unit Scaling 201
    Drawing Coordinate Axes 202
  Plotting Operations 207
    PENUP 207
    PEN 207
    PLOT 208
  Moving and Drawing 211
    MOVE 211
    DRAW 211
    Drawing Curves 212
    Padding the Increment Loop 213
    Problems 216
    IMOVE 217
    IDRAW 217
    Problem 220
  Labeling Graphs 221
    Label Direction 224
    Label Length 225
    Positioning Labels 227
    Problem 230
  INPUT in Graphics Mode 230
  Problem 237
  Advanced Plotting With BPLOT 237
    Procedure for Building the String 238
    Using the String With BPLOT 241
    Condensing the String Assignment Program 243
    Problem 253

Section 13: Debugging and Error Recovery 255
  Tracing Program Execution 255
    Tracing Branches 255
    Tracing the Values of Variables 255
    Tracing All Statements and Variable Assignments 256
    Cancelling Trace Operations 256
  The STEP Key 259
  Checking a Halted Program 260
  Error Testing and Recovery 260
  Some Hints About the System 262
  Memory Conservation Hints 263

Appendix A: Accessories 265

Appendix B: Owner's Information 269

Appendix C: Reference Tables 291

Appendix D: Glossary and BASIC Syntax Guidelines 295

Appendix E: Error Messages 303

Appendix F: Sample Solutions to Problems 309

Index 337

Meet the HP-85 Personal Computing System

Your Hewlett-Packard 85 Personal Computer is a versatile, self-contained, personal computing device which enables you to perform a wide variety of useful and interesting functions. To mention just a few of the special features of your HP-85, you have the ability to:

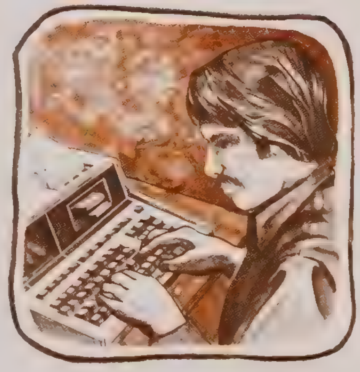
- Perform calculations in a simple, straightforward manner—as if you had a calculator with dozens of mathematical and scientific functions.
- Compose programs in BASIC (Beginner's All-Purpose Symbolic Instruction Code) programming language. The HP-85 exceeds the latest American National Standard for Minimal BASIC. In many areas, the HP-85 includes *enhancements* to this standard.
- Execute BASIC programs. After you have written your programs, they may be executed, often at the touch of one key. The HP-85 offers several *typing aids* to your program execution and control.
- Load and store programs and data on a magnetic tape cartridge with the *built-in* tape drive. Thus you may permanently store your programs to be retrieved again, whenever you wish.
- List programs and data with the *built-in* thermal printer. Not only can you list programs, but you can copy anything that appears on the display onto the printer, to record and review your results.
- Perform graphics. The graphics capabilities of the HP-85 are sophisticated, yet easy to use. And again, anything that you can ''draw'' on the display can be transformed to hard copy with a single command: COPY.
- Edit, correct, and modify anything that appears on the display with *tremendous* ease. In fact, the HP-85 allows you to access and review 64 lines of characters on the display, and to edit them at your convenience.

Now let's take a closer look at the HP-85 system to see how easy it is to use, whether we solve a problem manually, use one of the sophisticated prerecorded programs from the Standard Pac, or even write our own program.

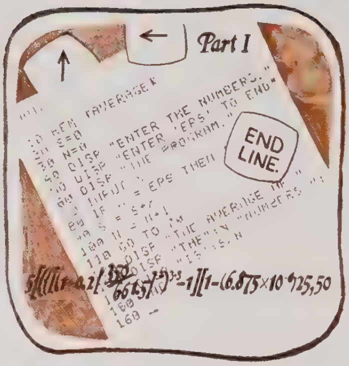
How to Use This Manual

This handbook has been designed to enable you to use the utmost potential of your HP-85 Personal Computer and to answer your questions concerning BASIC programming with the HP-85.

If you have just received your new HP-85 Personal Computer, read appendix B before you attempt to operate the system. Appendix B contains initial set-up instructions and other pertinent owner's information.

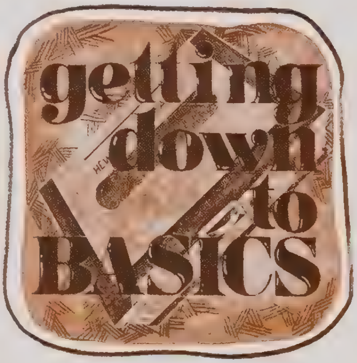


Then familarize yourself with the HP-85 system by reading and following through the examples in part I of this handbook—with your computer. The best way to feel at ease with the system is to sit down with the owner's handbook and the HP-85 and actually key in the examples provided in each of the sections. It won't take long to become familiar with the system, and it's well worth the time you invest to obtain a more complete understanding of your HP-85. Even if you are an advanced programmer, you will benefit from the unique features and capabilities of your HP-85 that are introduced in part I.

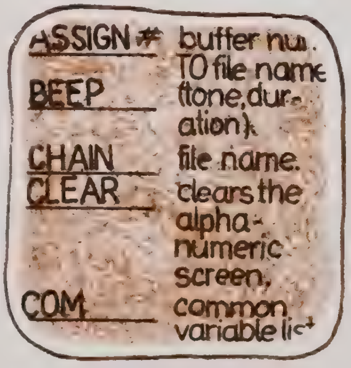

Part I
END LINE.

Part II of the *HP-85 Owner's Manual and Programming Guide* discusses each of the BASIC statements used with the HP-85. It also covers tape cartridge operations, graphics on your HP-85 system, and debugging procedures. There are problems for you to work at the end of most of the sections in part II and, in case you get stuck, sample solutions are given in appendix F.

If you are a beginning programmer, and you have difficulties with part II, you may wish to refer to the HP-85 BASIC Training Pac. The pac is designed to help you get acquainted with the HP-85 and BASIC programming.


getting down to BASICS

If you are an experienced programmer, you'll probably start programming with the HP-85 as soon as you've read part I. You can use part II as a reference guide to particular BASIC statements, but you'll probably find the *HP-85 Pocket Guide* and the *HP-85 BASIC Reference Card* most suited to your BASIC reference needs.


ASSIGN # buffer nu.
TO file name
BEEP (tone,duration).
CHAIN file name.
CLEAR clears the alpha-numeric screen.
COM common variable list

Where can you go next? After you've become familiar with the HP-85 itself, you may wish to enhance your programming capabilities with specific applications pacs, a memory module, extended capability ROMs, and peripherals. Be sure to check the accessories list in appendix A.

Travel case
RAMS
Appli-cation pacs
ROMS
Cartridges
Power cord
Thermal paper
Binders

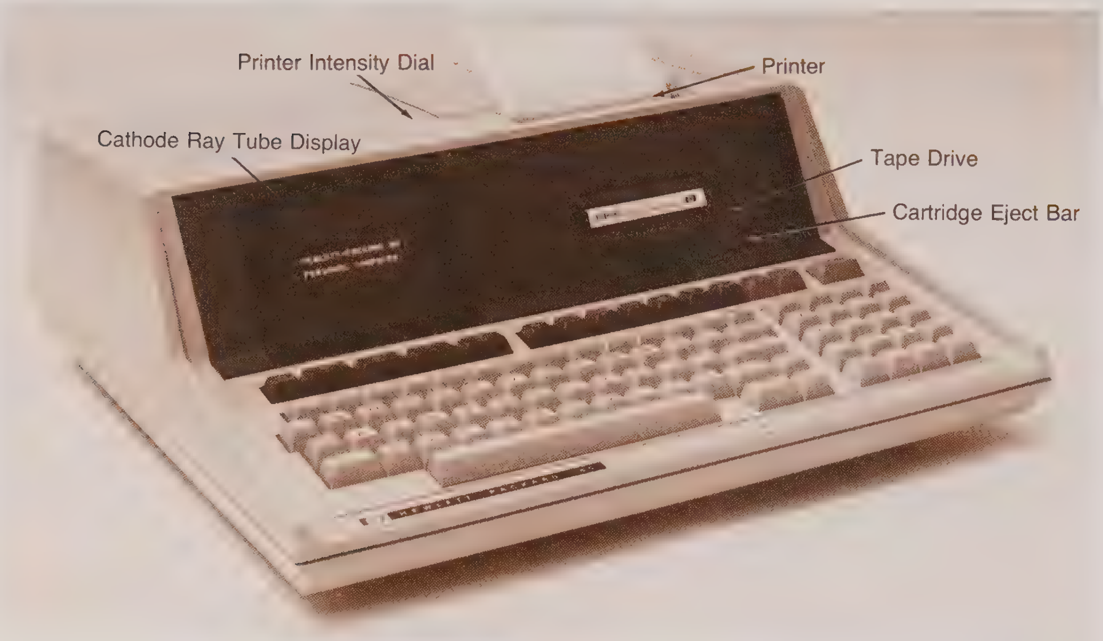
CAUTION

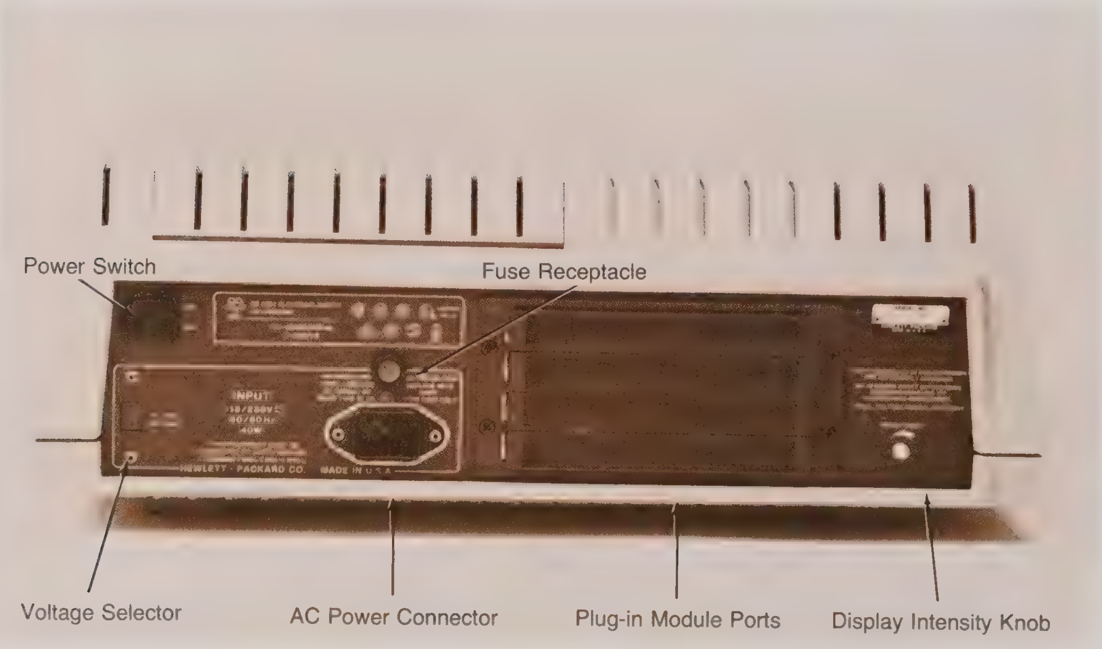
The inspection procedure and initial set-up instructions for the HP-85 are presented in appendix B of this manual. Please refer there:

- If you have not inspected the HP-85.
- If there is any doubt regarding the compatibility of the system power requirements to the available power in your area.

Do not attempt to set up the HP-85 without first becoming thoroughly familiar with appendix B; it contains information that is important to avoid damaging your personal computer when it is initially set up.

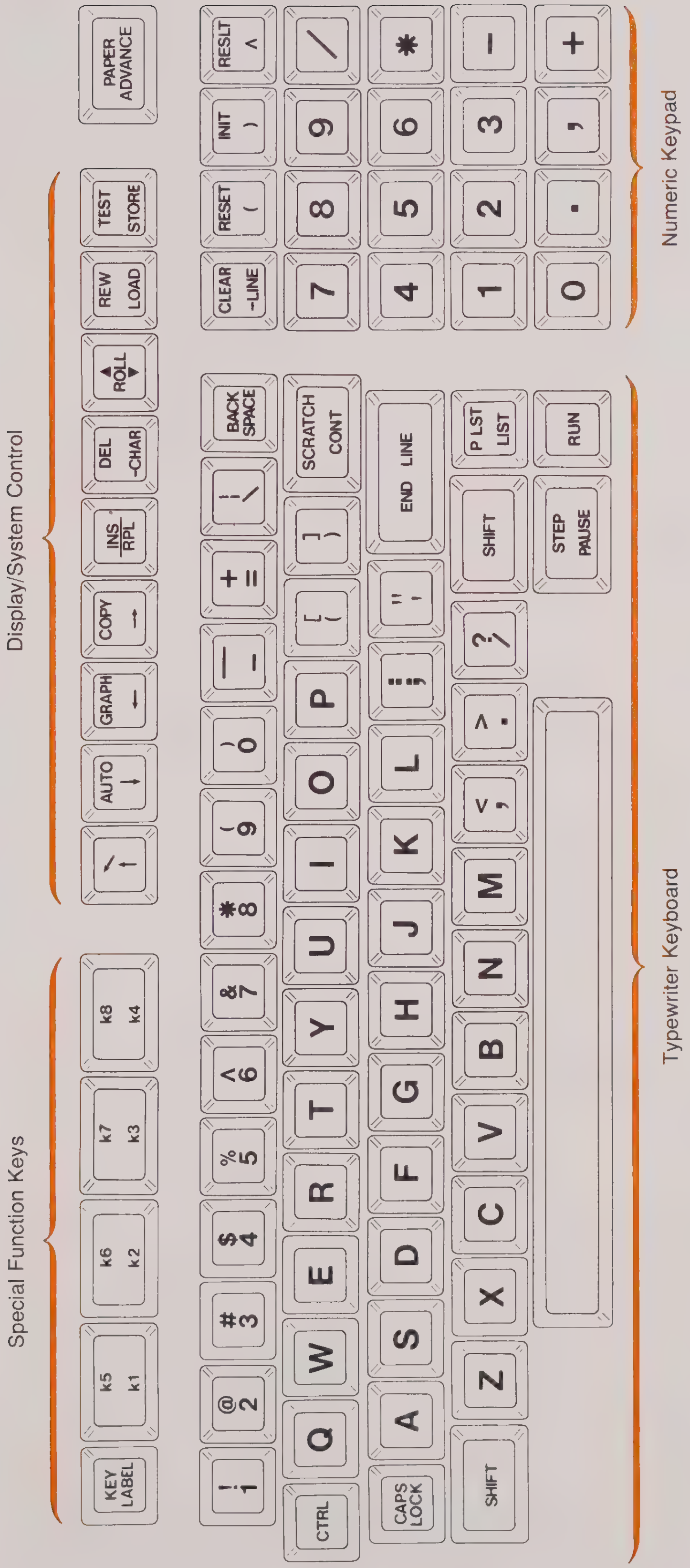
An Overview of the Hewlett-Packard 85


Printer Intensity Dial
Printer
Cathode Ray Tube Display
Tape Drive
Cartridge Eject Bar


Power Switch
Fuse Receptacle
Voltage Selector
AC Power Connector
Plug-in Module Ports
Display Intensity Knob

Back View

HP-85 Keyboard


Special Function Keys
Display/System Control
Typewriter Keyboard
Numeric Keypad
KEY LABEL
k5 k1
k6 k2
k7 k3
k8 k4
AUTO
GRAPH
COPY
INS RPL
DEL -CHAR
ROLL
REW LOAD
TEST STORE
PAPER ADVANCE
BACK SPACE
CTRL
SCRATCH CONT
CAPS LOCK
END LINE
SHIFT
P LST LIST
STEP PAUSE
RUN
CLEAR -LINE
RESET
INIT
RESLT

HP-85 Key Index

Typewriter Keys

A through Z. Alphabetic keys. In BASIC mode produce capital letters, and when used with SHIFT or CAPS LOCK produce small letters. In "typewriter" mode produce small letters, and when used with SHIFT or CAPS LOCK produce capital letters **(page 33).**

SHIFT Shift key. Used with the alphabetic keys to get reverse letter-case; with other keys to select alternate symbol, statement, or command on upper half of key **(page 33).**

CAPS LOCK Caps lock. Affects only the alphabetic keys. When pressed and locked, reverses the letter case of current typing mode **(page 33).**

CTRL Control. Used to select characters that are not normal typewriting characters and to output keycodes with decimal values below 32 **(page 34).**

Numerics, punctuation, symbols. The remainder of the typewriter keys operate like a standard typewriter. To select the symbol on the upper half of a key, hold SHIFT while you press the key **(page 33).**

END LINE Enters an expression, statement, or command into the computer to be interpreted and/or executed. Also performs a carriage return **(page 37).**

Numeric Keys

0 through 9 Digits. . Decimal point. Used for keying in numbers **(page 18).**

+ − * / ^ \\
Arithmetic operators: addition, subtraction, multiplication, division, exponentiation, and integer division, respectively **(page 43).**

( ) Parentheses. Used to key in numeric expressions and to enclose the arguments of functions **(page 45).**

, Comma. Separates input items and used as a separator in functions, statements, and commands **(page 47).**

RESLT Recalls to the display the most recently calculated result **(page 46).**

Special Function Keys

k1 through k4 (unshifted) and k5 through k8 (shifted). Special function keys for user-defined functions. Must be defined in a program **(page 154).**

KEY LABEL Recalls the current labels for the special function keys and displays them on the CRT. Also moves the cursor to the upper left corner of the display **(page 154).**

Display Control

↖ ↑ ↓ ← → Positions the cursor on the CRT display in the direction of the arrow, without erasing characters **(page 19).**

INS RPL Insert/Replace. Toggles between insert mode and replace mode. When the cursor is under a character in *replace* mode, typing any character will replace the character at the cursor position.

In *insert* mode, two cursors appear and the next character typed will be inserted between the characters marked by the cursor locations **(page 39).**

-CHAR Deletes the character above the cursor **(page 38).**

-LINE Deletes a line from the cursor to the end of the line **(page 19).**

BACK SPACE Erases characters as it backspaces **(page 38).**

SHIFT BACK SPACE Backspaces rapidly **(page 38).**

CLEAR Clears 16 lines of the display from the cursor position, then rolls the information above the cursor out of view and homes the cursor **(page 38).**

ROLL Recalls information that has "rolled" out of view. Pressing ROLL rolls down information that has most recently left the display, while SHIFT ROLL rolls up the oldest information saved on the display **(page 36).**

GRAPH Sets the system display to graphics mode, showing the current graphics display. Press any alphanumeric key to return to the normal alphanumeric display **(page 197).**

Cartridge Control

REW Rewinds the tape cartridge **(page 280).**

Program Control

AUTO Typing aid to display AUTO on the CRT display. The AUTO command instructs the computer to number program statements automatically. You may specify, at your option, the beginning line number and numbering interval; otherwise the system will number program lines beginning with 10 and incrementing by 10. The AUTO command is then executed by pressing END LINE **(page 80).**

DEL Typing aid to display DELETE on the CRT display. The DELETE command is used to delete a line or a section of a program. DELETE must be followed by the line number, or the first and last line number of a section of a program to be deleted. The DELETE command is then executed by pressing END LINE **(page 95).**

PAUSE Immediate execute key which halts a running program without otherwise affecting the program. Produces an audible beep when interrupting program execution **(page 97).**

CONT Immediate execute key used to continue execution of a program that has been halted by a PAUSE statement **(page 98).**

INIT Initializes (allocates memory to) a program without executing it **(page 99).**

STEP Executes a single program statement. The program must first be initialized by either RUN or INIT before you can single step through it **(page 259).**

RUN Immediate execute key which first initializes the current program, then executes it **(page 99).**

LOAD Typing aid to display LOAD on the CRT display. The LOAD command loads a specified file from a tape cartridge. LOAD must be followed by a file name within quotes or a string expression that specifies the file name. The command is then executed by pressing END LINE **(page 179).**

STORE Typing aid to display STORE on the CRT display. The STORE command stores a specified file onto a tape cartridge. STORE must be followed by a file name within quotes or a string expression that specifies the file name. The command is then executed by pressing END LINE **(page 176).**

LIST Immediate execute key which displays one full screen of the current program in memory starting at the beginning of a program. Each successive time LIST is pressed, another screen full of program lines is displayed until the end of the program is reached. Following the list of the last program line, LIST displays the remaining number of memory locations **(page 75).**

P LST Immediate execute key which will list the current program in its entirety on the system printer. Press any key to halt the printer listing **(page 75).**

Printer Control

PAPER ADV Moves the paper one line. If the key is held down, the paper advance will repeat until the key is released **(page 35).**

COPY Immediate execute key which copies the exact contents of the display onto the system printer **(page 35).**

System Commands

RESET Returns the computer to its condition at power on, except that programs are not erased **(page 40).**

TEST Performs a functional test of the processor and built-in peripherals **(page 39).**

SCRATCH Typing aid to display SCRATCH on the CRT display. The SCRATCH command clears computer memory. The command is executed by pressing END LINE **(page 78).**

Part I
Using Your HP-85

3
4
5
4

Section 1

Getting Started

In this section, we will discuss many topics in relatively few pages so that you can:

- Do a wide variety of calculations in just a few minutes.
- Begin using the editing capabilities of the computer.
- Use the tape cartridges.
- Begin programming.
- Have some fun!

It is our intent to "get on board" fast! For this reason, some of the more sophisticated concepts are greatly simplified or reserved for later sections.

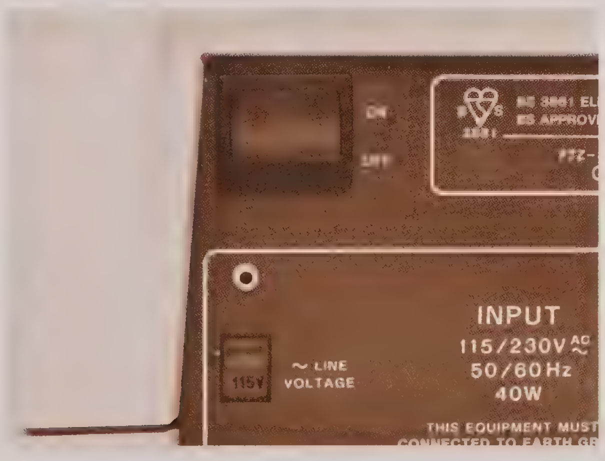
After working through this section, you'll have enough background to try things on your own, which is actually the best way to attain a good working knowledge of your personal computer. *And don't worry, you can't damage the HP-85 with any keyboard operation!*

Power On

There are a few things to notice each time you switch on your HP-85.

If the system is turned off:

- Set the power switch, located on the rear panel of the computer, to the ON position.



- When the cursor (underscore) appears after approximately 3 seconds in the upper left hand corner of the display (the "home" position), the HP-85 is ready to use.

- If a tape cartridge is present in the tape drive, the system will search for a program tape named "Autost" (for automatic start). The autostart routine permits the computer to load and run a program without operator instructions. More about this later.
- The system automatically runs through a self-test routine when the power is switched on. If it finds a problem in the system circuitry, it will beep and display:

  [illegible]

  This message means that your system is not operating properly; contact your local authorized dealer or your nearest HP sales and service office (addresses supplied in the back of this handbook).

  If the system is switched on and the tape drive is not being accessed, but the display remains blank, hold down the SHIFT key, then press RESET. This operation resets the system to a ready state (see page 40). Also adjust the display intensity knob on the rear panel of the system. If the display still remains blank, first check the power connection and the fuse, as described in appendix B. For further assistance, call your nearest HP dealer or HP sales and service office.

  If the system is on and the cursor is in the home position, you are ready to go!

  Before we begin, make sure that the CAPS LOCK key is released to the same level as the other keys.

Manual Problem Solving ("Calculator" Mode)

Let's try a few simple calculations to get the feel of your HP-85.

Type in the problems as you see them under the column marked **Press.** You may use either the numbers and arithmetic operators conveniently located on the right side of the keyboard or the numbers and symbols on the typewriter part of the keyboard. When you press END LINE, the answer will appear on the line below your input.

> **Note:** Any spacing that you use between characters, in manual calculations or in program statements, is totally arbitrary. When you list a program, the HP-85 adjusts the spacing of statements so that they can be output in their most legible form.

If you should make a mistake while typing the following problems, simply press the BACK SPACE key to erase the incorrect character, correct your mistake, and then continue keying in the problem.

To Solve	**Press**	**Display**	
5 + 6	5 + 6 END LINE	5 [illegible]	The result appears below the problem, indented one space.
9 × 8	9 * 8 END LINE	9 * 8	
2^9	2 ^ 9 END LINE	2 [illegible]	
$\sqrt{7921}$	SQR(7921) END LINE	[illegible]	
Sine of 3.3 radians	SIN(3.3) END LINE	SIN(3.3)	The system "wakes up" in radians mode. You can change it to degrees mode by typing DEG END LINE.

Arithmetic expressions are typed algebraically—just as you would write them on paper. Functions, like SQR and SIN, must be followed by the "argument" (or number) enclosed within parentheses. A complete list of functions may be found in appendix D. And, as you have seen, you must press END LINE to tell the system to solve the problem.

Simple Display Editing

Next, let's make some intentional errors and learn how to correct them. Suppose you wish to type the expression `3+INT(SQR(45*ABS(3-PI/.2)))`, but by mistake, type the following (don't press END LINE):

```
3+INT(SQR845*ABS
```

At this point, you realize that the `8` should have been a left parenthesis. The line could be corrected by backspacing and retyping. Let's save some typing time by pressing the ← *(cursor left)* key. The ← key enables you to backspace without erasing characters that are already on the display. Press the ← key until the cursor rests under the `8`. Then type `(`.

Now finish the problem by holding down the → *(cursor right)* key, until the cursor is past the `S` in `ABS`. Then type:

```
(3-PI/.2)))
```

The whole line should appear as:

```
3+INT(SQR(45*ABS(3-PI/.2)))
```

When you press END LINE, the answer will appear (`26`). Parentheses specify which operations are performed first—more about this in section 3.

As you may have guessed, just as the ← and → move the cursor back and forth on the display, the ↑ *(cursor up)* and ↓ *(cursor down)* keys move the cursor up and down on the CRT display. Thus, you can edit any line on the display (and more, as we shall see later). Finally, the ↖ *(home)* key returns the cursor to the home position on the display.

For example, using the ↑ key, move the cursor up the display so that it rests under the `S` in `SQR(7921)` in the problem that you solved above. Now, using the → key, move the cursor so that it rests under the `7`. Then type 980.

Now the line should read:

```
SQR(9801)
```

When you press END LINE, you will be finding the square root of the new number, 9801. The answer, `99`, will appear below the line you edited; the cursor will appear below the answer, ready for another problem.

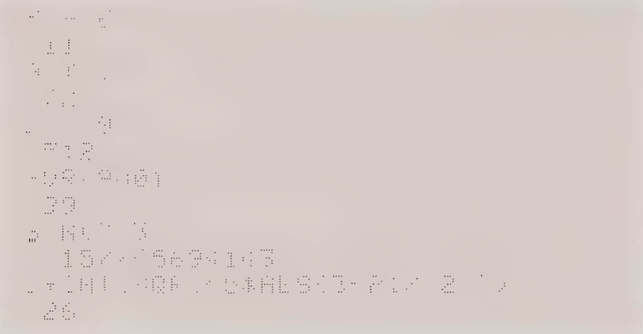
Note that you did not have to move the cursor past the right parentheses in the problem above before you pressed END LINE. The cursor may rest anywhere directly under the problem that you wish to enter into the computer. The HP-85 will read the full line, regardless of the cursor placement under the line.

Remember, *what you see* on the display, *is what you get*. If you have extra characters on the same line as you are editing, be sure to clear them before you press END LINE. You can erase characters to the left of the cursor by pressing BACK SPACE and erase characters to the right of the cursor by pressing the space bar or by pressing -LINE.

Clearing the Display

The -LINE *(clear to end of line)* key clears a line from the cursor to the end of the line. The SHIFT CLEAR keys clear the display and return the cursor to the home position. Typing `CLEAR` END LINE also clears the display.

If you have been following along with the examples, the display should look like this:



After editing and executing this line ...

... the cursor rests here.

If you press (END LINE) now, with the cursor resting under the S in SIN(3.3), you will be executing the function again. Instead, do the following:

Type	Display
5/4	5/4(3.3)

Here, you replaced the letters SIN, with 5/4. Before you can execute the expression you must clear the characters (3.3). You can do this by pressing the space bar until the cursor moves past the right parenthesis. But a faster and easier way to erase the characters is to press (-LINE).

Press	Display	
(-LINE)	5/4_	Characters from cursor to end of line are now deleted.
(END LINE)	1.25	Result.

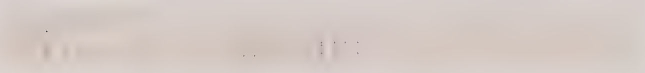
Why don't you press (SHIFT) (CLEAR) now, to clear the display before we continue?

Error Messages and Warnings

If you attempt an improper operation, the HP-85 beeps and displays the word Error or Warning, followed by a number and short description. The error number corresponds to a particular error condition that will help you pinpoint the error. A complete list and description of error messages is provided in appendix E.

There is no need to worry if the HP-85 returns an error or warning message—no keyboard operation is capable of damaging the system. Furthermore, most errors can be simply and easily corrected by editing the line in which the error occurred.

For example, executing the following expression will display an error message:



This expression is not complete because the right parenthesis has been left out. The system cannot interpret the expression so an error message is displayed and the cursor returns to the position in the expression where the system first detected an error—in this example, the asterisk. The cursor returns to the line you have executed when the system interprets an attempt to enter a program statement. (Actually, the system tries to interpret a line first as a program statement and then as an expression. If both attempts fail, the system reports the first error it finds.)

Now you may either edit and correct the line, or clear it by pressing (-LINE), or clear the whole display by pressing (SHIFT) (CLEAR).

Or you can forget about the error and use the arrow keys to position the cursor elsewhere on the display.

With most errors that occur during math calculations, the system displays a warning message and a default value. Then the cursor moves to the beginning of a new line.

For example,

```
5/0
Warning 8 : /ZERO
 9.99999999999E499
_
```

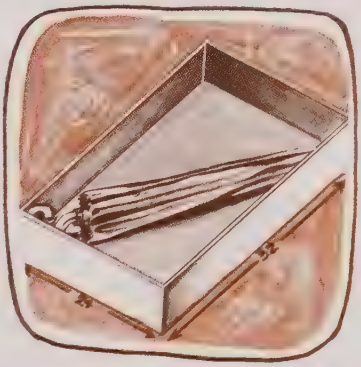
Division by zero causes a warning message and the default value to be displayed.
The cursor moves to the beginning of a new line.

Here, the system alerts you to the error, and then waits in a ready state for you to enter another expression.

Variables

Often it is convenient to assign values to letters and then use these letters in expressions. In programs, a letter can have its value continually updated or changed—hence the term "variable." But you can also perform variable arithmetic straight from the keyboard.

Example: Suppose you receive a telegram from your archaeologist friend, Arthur I. Factus, in South America. He's soon joining an expedition through the rain forest and writes, "PLEASE SEND UMBRELLA IMMEDIATELY." You find a shallow rectangular box, 24 inches wide by 32 inches long. What is the maximum length of an umbrella that will fit inside the box?



You can easily determine the diagonal length of the box, using the Pythagorean theorem, $d = \sqrt{l^2 + w^2}$, where d is the diagonal, l is the length of the box, and w is the width of the box.

One way to solve the problem is to type the following:

```
SQR(24^2+32^2)
```

Here, you substituted the dimensions of the box for the variables in the formula. When you press (END LINE), the answer, `40`, appears on the next line.

Another way to solve the problem is to assign the dimensions of the box to variable names, type in the formula, and let the HP-85 do the substituting. A variable name can be either a letter of the alphabet or a letter followed by a number 0 through 9.

First, ensure that the paper roll has been properly installed in the system, (refer to appendix B) and then type:

[illegible] (END LINE)

Note: To conserve power, the display turns off when the printer prints.

Then:

Press	Display		Printer
W = 24 (END LINE)	`W = 24`	Assigns width of box to W.	`W = [illegible]`
L = 32 (END LINE)	`L = 32`	Assigns length of box to L.	`L = 32`
D=SQR(W∧2+L∧2) (END LINE)	`D=SQR(W^2+L^2)`	Evaluates expression and assigns a value to D.	`D=SQR(W^2+L^2)`
D (END LINE)	`D` [illegible]	Fetches the value of D and displays it.	`D` [illegible]

You can assign a numeric value or the result of an expression to a variable name, as shown above. Whenever you wish to recall the value of an assigned variable, type the variable name and press (END LINE). (Although it may be extra work for this problem, variables are extremely useful in programs in which the values of variables are always changing.)

And you can see the printer has preserved a record of your calculation. Press the paper advance, located in the upper right-hand corner of the keyboard, and save this printout. You are going to use it to write a BASIC program for the HP-85. But first let's look at a prerecorded program—one of the 15 that are included with the Standard Pac shipped with your computer.

With the system in print all mode, you will have a printed copy of everything that you type and that the computer displays. If you wish to cancel print all mode, type:

[illegible] (END LINE)

The [illegible] command returns the system from print all mode to normal display mode.

Running a Prerecorded Program

The Standard Pac magnetic tape cartridge shipped with your HP-85 contains 15 prerecorded programs. By using programs from the Standard Pac (or from any of the optional application pacs available in areas like finance, statistics, mathematics, engineering, linear programming, beginning BASIC programming, ...) you can use your HP-85 to perform extremely complex computations just by following the directions in each pac. Let's try running one of these programs now.

Loading a Program From the Standard Pac

1. Before you insert the Standard Pac tape cartridge, make sure that the RECORD slide tab is in the left-most position (as shown). This will protect your tape, so that no other programs can be accidently recorded on the tape.

STANDARD PAC
00085-13001A

When the RECORD slide tab is in the left-most position (the opposite direction of the arrow), nothing can be recorded on the tape; your tape is protected.

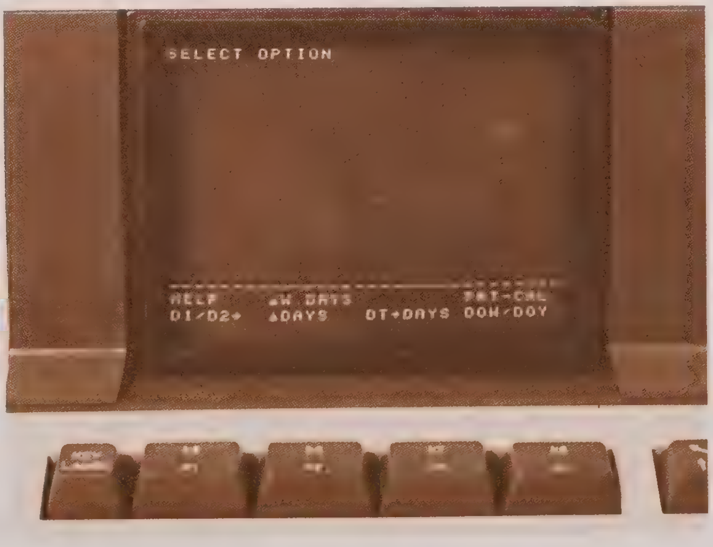
2. Insert the tape cartridge so that its label is up and the open edge is toward the computer. The tape drive door will open when the cartridge is pressed against it; the cartridge can then be inserted. (To remove the tape cartridge, you must press the eject bar. If it is pulled out without pressing the eject bar, another cartridge cannot be inserted until the eject bar is pressed.)

3. To load the Calendar Functions program, type:

   (LOAD) "CALEND" (END LINE)

   The LOAD command instructs the computer to find the specified program on the tape cartridge and then load it into computer memory. The CRT screen will blank out while the HP-85 is searching for and loading the program. And an amber light, located to the left of the eject bar, will glow while the cartridge is being used to let you know that the system's attention has been transferred to the tape drive.

4. When the cursor returns to the display and the amber tape drive light goes out, press the (RUN) key to start the program. After you press (RUN), the following should appear on the display:



```
SELECT OPTION



-------------------------------------------
HELP      ΔW DAYS            PRT-CAL
D1/D2→    ΔDAYS    DT→DAYS   DOW/DOY
```

Many of the programs in the Standard Pac use the special function keys. This is how they work. The bottom two lines of the display correspond directly with the special function keys on the keyboard. The bottom line of the screen displays the labels for the unshifted keys, k1 through k4; the line above it refers to the shifted keys, k5 through k8.

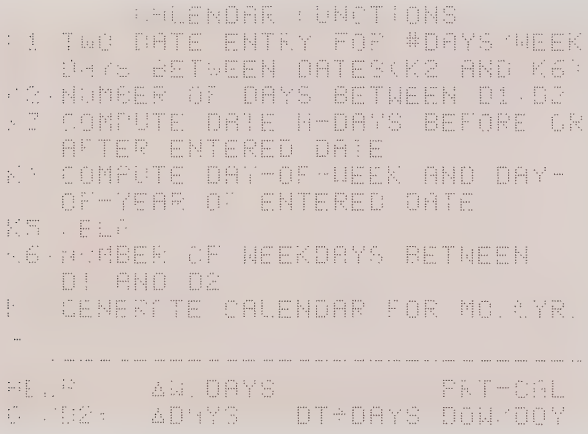
The key labels will remain on the display until they are over-written by characters that you type or that the HP-85 displays. In any case, you can always display the current labels on the screen by pressing KEY LABEL.

With `SELECT OPTION` and the key labels displayed, you are ready to use the program.

Example: How many days are there between November 25, 1945, and July 25, 1954?

Solution: Since this may be the first time you've used the Calendar Functions program, let's ask for help.

For a more detailed explanation of the key functions, press SHIFT k5.



```
         CALENDAR FUNCTIONS
K1 TWO DATE ENTRY FOR #DAYS/WEEK
   DAYS BETWEEN DATES(K2 AND K6)
K2 NUMBER OF DAYS BETWEEN D1,D2
K3 COMPUTE DATE N-DAYS BEFORE OR
   AFTER ENTERED DATE
K4 COMPUTE DAY-OF-WEEK AND DAY-
   OF-YEAR OF ENTERED DATE
K5 HELP
K6 NUMBER OF WEEKDAYS BETWEEN
   D1 AND D2
K8 GENERATE CALENDAR FOR MO.&YR.

HELP     ΔW.DAYS          PRT-CAL
(D1/D2)  ΔDAYS   DT+DAYS  DOW/DOY
```

Use k1 (D1/D2→) to enter the two dates.
Then use k2 (Δ DAYS) to find the number of days between the dates.

k7 is not used in the Calendar program.

If you wish to have a printed copy of the display as you see it, simply press SHIFT COPY.

Now let's continue—enter the dates:

Press	Display
k1	`ENTER FIRST DATE MM.DDYYYY?`

This means you key in the date in the form: month (01 to 12), decimal point, day (01 to 31), year (four digits). Press END LINE after you type the date to enter the data into the program. Thus, to enter November 25, 1945, and July 25, 1954:

Press	Display
11.251945 END LINE	`11.251945` `ENTER SECOND DATE MM.DDYYYY?`
07.251954 END LINE	`07.251954` `DATE ENTERED`

Now that the dates have been entered, press (k2) (ΔDAYS) to find the number of days between the dates:

Press

(k2)

Display

```
NUMBER OF DAYS BETWEEN
11.251945 AND 7.251954 IS
 3164 DAYS
```

With the HP-85 finding days between dates is that easy! Should you run into difficulties using the Calendar Functions program, refer to the user instructions in the Standard Pac.

Before we leave the calendar program for you to explore on your own, let's use the program to generate a calendar for January 1980 and demonstrate just some of the graphics capabilities of your HP-85. First clear the display.

Press

(SHIFT) (k8)
1,1980 (END LINE)

Display

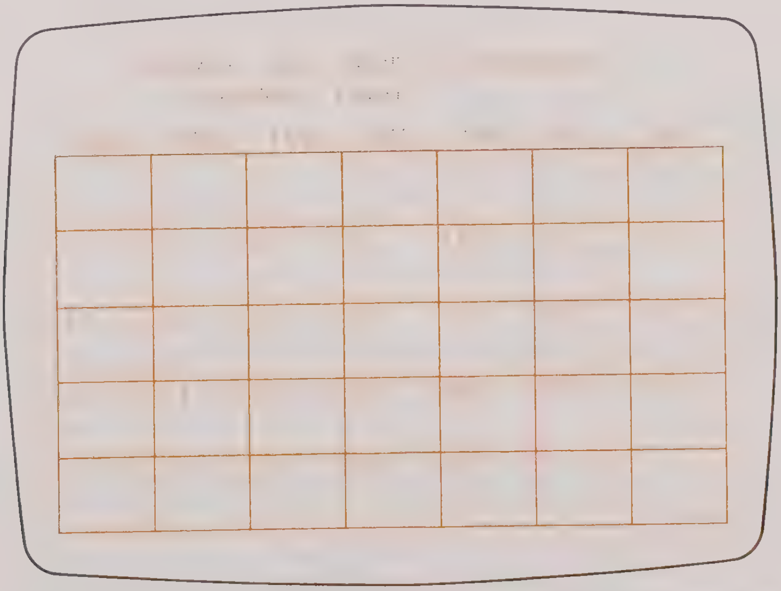
```
MONTH, YEAR ?
1,1980
ENTER HEADING?
_
```

Function key (k8) (PRT-CAL) requests a numeric entry for the month (1-12) and year. Again, enter data into the computer using (END LINE).

Here's your chance to be creative! You can type anything that will fit on one line. If you want your heading to look like ours, type:

```
HAPPY NEW YEAR!!!?%#&*+@!
```
(END LINE)

Now, watch the HP-85 go to work as it first "draws" the calendar on the video display and then copies it onto the printer. The display will blank out while the printer is in operation. When the printer has finished, you will have a printed copy of the calendar that appears on the display:



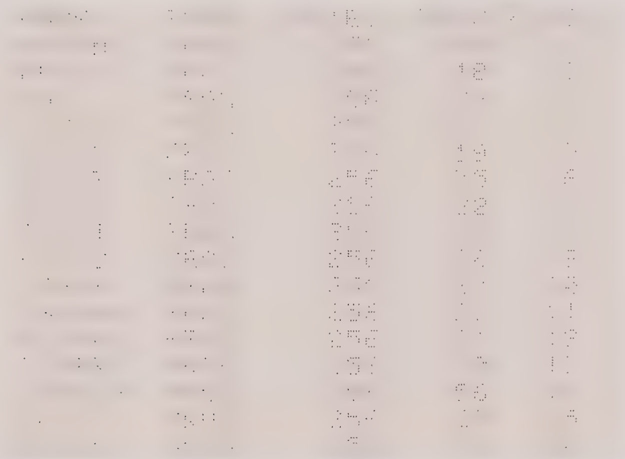
You may want to know the names of the other programs on the Standard Pac tape cartridge. But before you can view the tape directory (catalogue of programs on the tape), you must stop the calendar program from running.

Halting Program Execution

Press the PAUSE key to halt a running program at any time and return system control from the program to the user. (The system beeps when PAUSE is pressed and the program is running.) You may resume the execution of a paused program by pressing CONT *(continue)*.

Now lets take a quick look at the catalogue of programs on the Standard Pac tape cartridge. First press PAUSE to stop the calendar program, then type:

```
[illegible] END LINE
[illegible] END LINE
```



Return the system to normal display mode again by typing [illegible] END LINE.

The [illegible] *(catalog)* command returns the names of the programs on the tape along with some other information about the files. We will discuss the tape filing system in more detail in section 11. For now, just note the names of the programs in the left-most column.

You have seen from the calendar functions example how simple and how much fun it is to use your HP-85. You can run the program again as often as you like. And you can begin using your Standard Pac, or any of the optional application pacs, right *now*. Load any program stored on tape using the `LOAD` command, followed by the program name in quotes. All you have to do to begin taking advantage of the computing power and programmability of the HP-85 is follow simple instructions like these.

Writing Your Own Programs

If you have never written a program, you may possibly feel uneasy about programming. No need to worry! BASIC is easy to use, yet enables you to perform many complex operations.

BASIC makes use of statements that resemble English. Once a statement is explained, its function is easy to remember. A BASIC program is made up of numbered statements which direct the system to perform certain tasks.

Earlier, you calculated the diagonal of one side of a rectangular box, and you may have saved the printed copy with the values and formula for the problem. Now, if you want to calculate the diagonal of several rectangles (or the hypotenuse of any right triangle), you could repeat the procedure, using different values for the dimensions of the sides. Or you could change the values of the variables using the editing capabilities of the HP-85.

The easiest and fastest method, however, is to create a BASIC program that will compute the diagonal of any rectangle.

Creating the Program

Essentially, you have already created it. When you write a program, you must ask yourself the following questions:

1. What answer(s) do I want?
2. What information do I know?
3. What method will I use to find the solution from what I know?
4. How can the HP-85 help me solve the problem?

We want to find the diagonal of any rectangle. We know that we can use the Pythagorean theorem to compute the diagonal given the lengths of the sides of the rectangle. Thus, we know that we must assign values to two variables and then compute the result using the given formula. We'll answer the other two questions below (and discuss the details of BASIC programming in part II of this handbook).

For now, notice that each statement begins with a number and the last statement of a program is `END`. (You may wish to clear the display before you key in the program; press SHIFT CLEAR.)

Entering the Program

To enter the program into the system:

1. Press SHIFT SCRATCH and then press END LINE to clear the computer and erase previous programs from computer memory.
2. Type the following program exactly as shown (including the statement numbers), pressing END LINE after each statement.

```
[illegible] DISP "ENTER SIDE LENGTHS"   END LINE
[illegible] DISP "OF A RIGHT TRIANGLE"  END LINE
[illegible] DISP "SEPARATED BY A COMMA" END LINE
[illegible] DISP "THEN PRESS END LINE"  END LINE
[illegible] [illegible] END LINE
[illegible] INPUT L,W END LINE
[illegible] END LINE
[illegible] END LINE
[illegible] END LINE
```

Statements 10 through 40 display the quoted text on the CRT screen.

Audio, as well as visual, prompt!

Enables you to assign values to L and W from the keyboard.

Computes the hypotenuse.

Prints quoted message and value of D.

Marks end of program.

Running the Program

To run the program, simply press the RUN key. Find the length of the hypotenuse of a right triangle with sides 7.5 inches and 10 inches.

Press	Display
RUN	[illegible]
	Beep!
7.5,10 END LINE	[illegible]

Now the HP-85 will print the result:

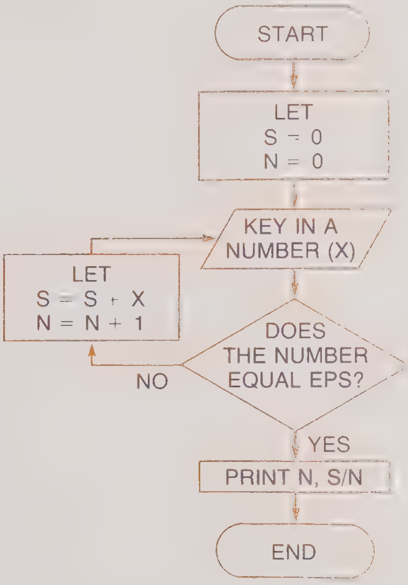
```
[illegible] = 10.5
```

You can run the program as many times as you like, simply by pressing the (RUN) key.

An Averaging Program

Since you may not be sending umbrellas to South America in the near future, or calculating the hypotenuses of right triangles or the diagonals of rectangles, let's write a program that may be of more use to you, and then record it on a tape cartridge.

This flowchart outlines the steps in a program that enable you to enter a set of numbers and then find their average.



First, we initialize the variables we will use. S will be the sum of the numbers, N determines how many numbers are being averaged, and X represents each new number.

When you key in EPS (which stands for epsilon, the smallest number you can obtain on the HP-85), the program stops asking for new numbers and prints the average of the numbers you have keyed in.

Before you key in the following program, be sure to press (SHIFT) (SCRATCH) (END LINE) to erase the previous program.

And let's use the (AUTO) key to provide us with statement numbers automatically, so that we don't have to type them ourselves. Simply press (SHIFT) (AUTO) and the system will display:

Then press (END LINE) and the system will display 10 and wait for you to enter a program statement. After you enter the statement by pressing (END LINE), the system will display 20 and wait for another statement to be entered. The AUTO command numbers statements beginning with 10 and in increments of 10. (You can also change the starting number and increment value with the AUTO command, as we shall see later.) Stop auto line numbering by backspacing over the unwanted statement numbers and typing (END LINE).

Now enter the averaging program below. From now on, we won't be showing the END LINE key with the program listing, but, **it must be pressed after each statement.** When the system displays the statement number, enter the rest of the statement and press END LINE.

```
AUTO

10 REM *AVERAGE*
20 S=0
30 N=0
40 DISP "ENTER THE NUMBERS."
50 DISP "ENTER 'EPS' TO END"
60 DISP "THE PROGRAM."
70 INPUT X
80 IF X = EPS THEN 120
90 S = S+X
100 N = N+1
110 GOTO 70
120 DISP "THE AVERAGE OF ";
130 DISP "THE";N;"NUMBERS ";
140 DISP "IS";S/N
150 END
160 _
```

Press SHIFT AUTO END LINE for automatic line numbering.
Remark.
Initialize variable S.
Initialize variable N.
Display quoted message.

Assign a value to X from the keyboard.
Check X to see if it is EPS.
Add X to sum.
Add 1 to counter.
Go back to line 70 to enter a new number.
Display result.

Marks end of program.
Now backspace over `160` and type `NORMAL` END LINE to stop auto line numbering.

Let's take an example to test the program.

Example: What is the average of the distances in light-years of the five brightest stars (aside from the sun) seen from the earth?

Star	Distance (light-years)
Sirius	8.7
Canopus	100
Alpha Centauri (Rigil Kentaurus)	4.4
Arcturus	36
Vega	26.5

Press	Display

RUN	`ENTER THE NUMBERS.`
	`ENTER 'EPS' TO END`
	`THE PROGRAM.`
	`?`

8.7 END LINE	`8.7` `?`
100 END LINE	`100` [illegible]
4.4 END LINE	[illegible]
36 END LINE	[illegible]
26.5 END LINE	[illegible]
EPS END LINE	[illegible]

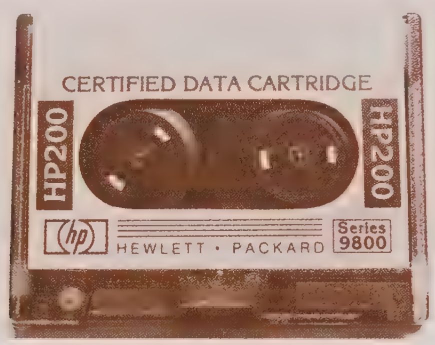
Recording the Program

Just as the programs in the Standard Pac have been recorded on a magnetic tape cartridge, you also can record your own programs on a cartridge.

To Record Your Program:

1. Select a blank magnetic tape cartridge. Use only HP Data Cartridges with your HP-85.

2. Check to see that the RECORD slide tab is in the right-most position, in the direction of the arrow (as shown). When the RECORD slide tab is in the left-most position, your cartridge is protected; i.e., you cannot record anything on it or delete from it.



When the RECORD slide tab is in the right-most position, you can record programs on the tape or delete existing programs from the tape.

3. Insert the cartridge so that its label is up and the open edge is toward the computer. (If the Standard Pac is still in the tape drive, take it out by pressing the eject bar.)

Note: The `ERASETAPE` command should only be performed the first time you use a new tape or when you wish to erase all of the existing programs on a tape.

4. Type `ERASETAPE` (END LINE). This command erases the tape and in the process, sets up a "directory" so that your programs can be filed.

5. Now, decide what you want to name your program. Pick something that will remind you of the program—a name no longer than six characters. However, any combination of characters may be used, except quotation marks. Then press the (STORE) key and type the name of your program enclosed within quotation marks. If you want your program to be stored like ours, type:

(STORE) [illegible] (END LINE)

When you press (END LINE), the HP-85 records your averaging program on the magnetic tape cartridge in its own "file." Again, notice that the CRT display is turned off to conserve power while the cartridge is accessed.

That's all there is to it! To record future programs on the same tape cartridge, you simply follow step 5 again. Future programs will automatically be stored in separate files.

You can verify that your program has been stored by executing the `CAT` command as we did earlier. The information displayed will be discussed later.

Erasing a Program From the Tape Cartridge

An averaging program may be of no use to you, so before we go on to the next section we'll tell you how to erase a specific program file using the `PURGE` command. First, make sure that the RECORD slide tab is in the right-most position. Then:

1. Type `PURGE`.
2. Type the name of the program or file you wish to delete from the cartridge, enclosed within quotation marks.
3. Press END LINE to purge the specified file.

If "AVERAG" was the name you used for the averaging program, simply type:

```
PURGE "AVERAG" END LINE
```

Your program or file will be erased, ready for something else to be stored there. You can store many long programs on one tape, so you don't have to purge little-used programs constantly.

Section 2

Keyboard, Printer, and Display Control

Now that you've had a chance to familiarize yourself with the HP-85, let's look at some of its features in greater detail.

The Keyboard

As you've noticed, the keyboard is divided into the following areas:

- Typewriter Keyboard
- Numeric Keypad
- Special Function Keys
- Display Control and System Command Keys

Some of the features in each area were discussed in section 1. The rest of the display editing features will be discussed in this section. The remaining keys are helpful in a variety of ways—as typing aids, in running programs, using the printer, and recording programs on tape. The keys are described, in appropriate places, throughout this manual. Refer to the HP-85 key index on pages 12 and 13.

Typewriter Keys

The alphanumeric keys operate much like those on a standard typewriter keyboard. If, for instance, you want to display the dollar sign, $, you must hold down the SHIFT key while you press $ 4. You must also use the SHIFT key to select any command or symbol on the upper half of a key. But we won't be showing the SHIFT key in the keystroke sequences in this handbook.

If the command is a shifted operation, it will appear in the upper half of the key. For instance, when you see CLEAR, it means you must hold the SHIFT key down while you press CLEAR -LINE.

The HP-85 keyboard differs from a standard typewriter in two major ways:

- Unshifted letters appear as capital letters on the display (unless you use the FLIP command, as we'll see in a moment).
- All of the keys repeat automatically if you continue to hold them down.

BASIC Typewriter Mode

Unshifted letters initially appear as capitals on the display because the standard BASIC language requires its "keywords" (like PRINT, GOTO, IF... THEN, etc.) to be in capital letters.

In BASIC mode you can select small letters by using the SHIFT or CAPS LOCK keys with the alphabetic keys.

Thus, when you press A, a capital "A" appears on the display; when you press A while holding down the SHIFT key, a small "a" appears on the display.

The CAPS LOCK key operates like the CAPS LOCK on a standard typewriter except that if the key is pressed and locked in BASIC mode, alphabetics appear as small letters. Once the CAPS LOCK key is pressed, it remains locked until you press it again. Note that *only* the 26 letters of the alphabet are affected by the CAPS LOCK key.

Normal Typewriting Mode

If you wish to type in ''normal typewriting mode'' where unshifted letters produce small letters and shifted letters produce capital letters, use the system command FLIP. Whenever you type FLIP and press END LINE, the unshifted case switches from small letters to capital letters or vice versa.

Programming Note: Even though standard BASIC requires "keywords" to be in capital letters, the HP-85 will interpret keywords and variables that are typed in either uppercase or lowercase letters. Thus, the following program, typed in normal typewriting mode, is legal:

```
10 print "You can use small"
20 print "letters or capital"
30 print "letters in BASIC"
40 print "programs."
50 end
```

As soon as you list the program, the small letters in keywords and variable names are converted to capital letters, but strings (quoted text) and remarks will remain as typed.

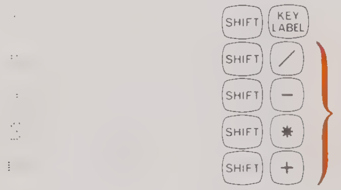
HP-85 Character Set

The HP-85 character set consists of 256 characters, 128 of which are directly accessible from the keyboard.

You can see the uppercase letters, punctuation and other typewriter symbols on the face of the keys; and you've seen how lowercase letters can be accessed using SHIFT, CAPS LOCK, or the FLIP command.

Five more characters can be accessed with the SHIFT key. Thirty-two more characters are accessed with the CTRL key. The remaining 128 characters can be accessed with the CHR$ function; they are merely the first 128 characters underscored.

The extra shifted characters are:



To access these characters, use operators from the numeric keypad only.

The CTRL characters are those in the first column of the table of characters in appendix C, page 292. They are generated by holding down the CTRL key and pressing the key that is superscripted by a ''c'' next to the character in the table.

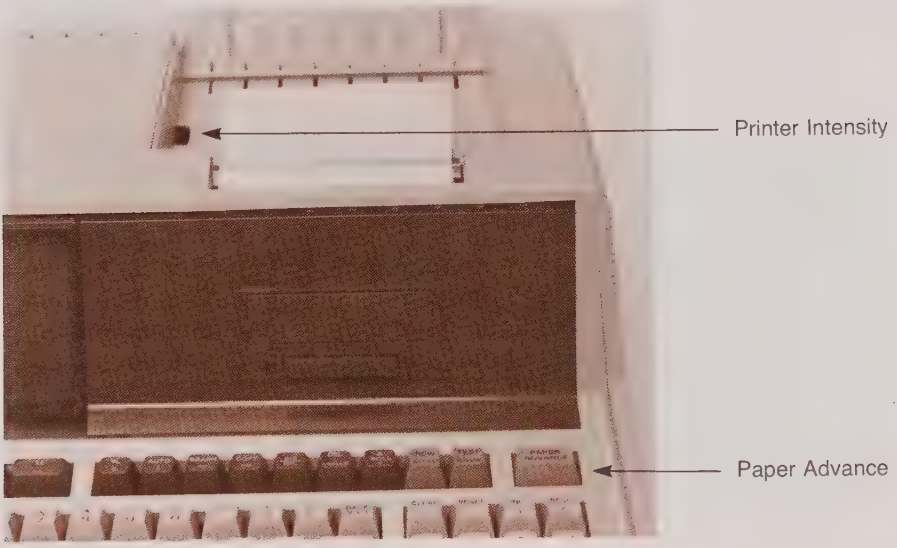
For instance, to generate the Greek letter Δ, hold down the CTRL key and press H (H^c).

And, since @ is a shifted symbol, generate the character [illegible] by holding down the CTRL key and SHIFT key and pressing @ 2 ($@^c$).

Each of the characters is assigned a decimal code, from 0 through 255. These codes are useful in advanced programming. We'll discuss character codes in section 8, Using Variables.

Printer Control

The HP-85's built-in thermal printer prints 32 characters per line.



Adjust the intensity of printed characters by rotating the printer intensity dial, located to the left of the paper roll. The lightest setting is when the dial shows 0, the darkest setting is when the dial shows 7. You can extend the long-term life of the printer by setting the printer intensity dial to 4 or less.

There are several ways to access the printer:

- Pressing the COPY key produces a printed copy of whatever is currently displayed on the CRT screen. The COPY key can be pressed to copy either the alphabetics or graphics on the display. You can also copy the alpha screen by *typing:*

  `COPY` END LINE

- Executing the `PRINT ALL` command sets the HP-85 to print all mode; everything that you enter into the system and every message or result that the system displays will be recorded by the printer.

  `PRINT ALL` END LINE

  Return to normal display mode by typing:

  `NORMAL` END LINE

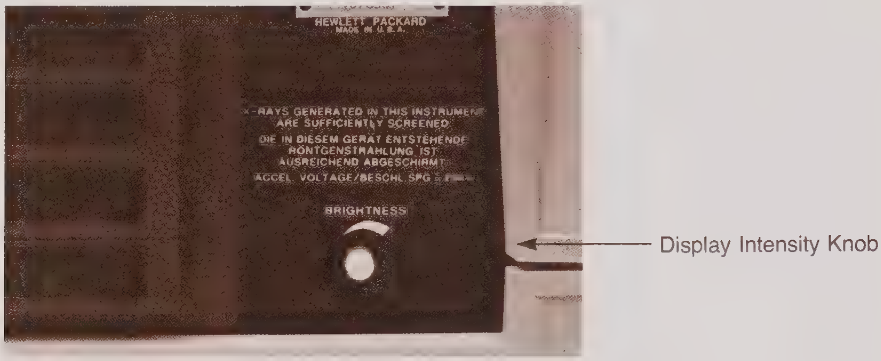
- And, of course, whenever you execute the `PRINT` statement, either manually from the keyboard or in a program, the `PRINT` message will be output to the printer.

To advance the printer paper, press the PAPER ADV key, located in the upper right corner of the keyboard. To advance the paper more than one line, simply hold the PAPER ADV key down until the paper has advanced the desired amount. To replace the paper roll, refer to appendix B.

The Display

The CRT (cathode ray tube) display consists of a 32-character by 16-line display screen and is the primary means of editing programs, and of viewing data, keyboard entries, program listings, error messages, system comments, and results.

You can increase the intensity of characters on the display by rotating the display intensity knob in the direction of increasing width of the brightness symbol.


HEWLETT PACKARD
MADE IN U.S.A.
X-RAYS GENERATED IN THIS INSTRUMENT ARE SUFFICIENTLY SCREENED
DIE IN DIESEM GERÄT ENTSTEHENDE RÖNTGENSTRAHLUNG IST AUSREICHEND ABGESCHIRMT
ACCEL. VOLTAGE/BESCHL. SPG.
BRIGHTNESS
Display Intensity Knob

You can display a maximum of 16 lines at any one time, but you actually have immediate access to four full screens worth (64 lines) of information.

The ROLL key is used to recall information that has ''rolled'' out of view. There are three full screens of past history, plus the current screen, available for rolling up or down. You'll appreciate the ROLL key when you are writing, reviewing, or listing lengthy programs.

When you hold down the ROLL key, information in the display will ''roll down'' to reveal the lines most recently lost.

When you press SHIFT ROLL, information in the display will ''roll up'' to reveal either the oldest lines (if no previous rolling has been done) or lines that have been rolled down (if some previous rolling has been done).

Entering Long Expressions

Suppose you wish to solve a lengthy numeric expression like:

$$\sqrt{5\left[\left(\left\{\left[\left(1+0.2\left[\frac{350}{661.5}\right]^{2}\right)^{3.5}-1\right]\left[1-(6.875\times10^{-6})\,25{,}500\right]^{-5.2656}\right\}+1\right)^{0.286}-1\right]}$$

Do you have to break the expression into parts and solve one line's worth of the problem at a time?

No! An expression can contain as many as 95 characters (including spaces) or three full lines of the display minus one character position for END LINE.

Before we attempt to evaluate the long expression, press one of the character keys, such as the * key, and continue to hold it down until it repeats across the display.

```
********************************
********************************
********************************
********************************
```

As long as you hold down the * key, row after row of asterisks will be repeated across the display. There is no need to press END LINE at the end of the line on the display; when the cursor reaches the end of a line, typing another character automatically sends it to the beginning of the next line.

Now press CLEAR to clear the display. As you type in the following expression, notice that when the cursor is at the end of a line, typing automatically sends the cursor to the next line. Don't press END LINE until you have keyed in the entire expression.

```
SQR(5*(((((1+.2*(350/661.5)^2)^3
.5-1)*(1-6.875E-6*25500)^-5.2656
)+1)^.286-1))  END LINE
```

Typing merely continues on the next line.

Now press END LINE to execute this expression.

```
  835724535179
```

The answer.

The 95-character maximum length of an expression also applies to program statements (including line numbers). For instance, in the Pythagorean theorem program in section 1 we typed:

```
10 DISP "ENTER SIDE LENGTHS"  END LINE
20 DISP "OF A RIGHT TRIANGLE,"  END LINE
30 DISP "SEPARATED BY A COMMA."  END LINE
40 DISP "THEN PRESS ENDLINE."  END LINE
```

But we could have entered the display message in one statement, like this:

```
10 DISP "ENTER SIDE LENGTHS OF A
 RIGHT TRIANGLE, SEPARATED BY A
COMMA. THEN PRESS ENDLINE."  END LINE
```

Again, at the end of a line on the screen, the cursor automatically moves to the beginning of the next line. But you must press END LINE to enter the program statement into computer memory. END LINE marks the end of an expression or statement *and* positions the cursor at the beginning of a new line.

What happens when you fill the display with characters, or type more than 95 characters in an expression or statement? The HP-85 will allow you to key in four full screens worth of characters as long as you don't press END LINE.

But, if you try to enter an expression consisting of more than 95 characters by pressing END LINE, you will probably get odd results. The system will try to interpret the most recently typed three lines of the display, yielding either an error message or interpreting only part of what you keyed in. If you are confused, execute the `PRINT ALL` command and the system will echo exactly what it understood your line to be.

Display Editing

In section 1, we introduced the following display editing features of your HP-85:

BACK SPAC
SCRATCH
CONT
END LINE
O
P
L

←	Cursor Left	These keys merely position the cursor in the display without erasing characters. The vertical and horizontal arrow keys repeat automatically if you continue to hold them down.
→	Cursor Right	
↑	Cursor Up	
↓	Cursor Down	
↖	Home	
CLEAR		Clears the display.
-LINE		Deletes a line from the cursor to the end of a line.
SPACE BAR		The space bar moves the cursor forward one space, or, if held down, repeats automatically. If characters are already present on a line when you press the space bar, they will be replaced with spaces.
BACK SPACE		Erases characters as you backspace. The key repeats when held down continuously.

There are three more important display editing features: SHIFT BACK SPACE *(fast backspace)*, -CHAR *(delete character)*, INS RPL *(insert/replace)*.

Fast Backspace

If you press both the SHIFT key and the BACK SPACE key at the same time, the cursor will *rapidly* backspace, erasing characters at the same time. To protect the user from accidently erasing the whole screeen, SHIFT BACK SPACE moves the cursor back to the beginning of a line, not to the home position of the display. But if you continue to hold down SHIFT BACK SPACE, it will repeat rapidly, erasing the next line above.

Deleting Characters

The -CHAR key enables you to delete a character from the display, without leaving a space in its place. If you hold down the -CHAR key, it repeats automatically.

Example: Type, without pressing END LINE:

This line will be durlete

But what we meant, of course, is that soon we will *delete* This line. Move the cursor with the ← key, so that it resets under the r and press -CHAR once.

Now move the cursor back to the beginning of the sentence, again with the ← key. Then hold down the -CHAR key to delete This line. The remainder of the sentence should look like this:

will be delete

And this can be deleted with the stroke of one key. Press -LINE to delete the rest of the sentence.

Example: Change:

[illegible]

to: 78 [illegible]

Position the cursor under the plus sign between 100 and 91, in the first expression, then press -CHAR until [illegible] 39 has been deleted. Now press END LINE to get a result of [illegible].

Inserting Characters

When there is only one cursor on the display, the computer is in *replace* mode. In other words, when you type characters "on top of" characters that are already in the display, those characters are replaced by the ones you type in.

The (INS RPL) *(insert/replace)* key alternates between *insert* and *replace* mode, allowing you to insert characters in a line that has already been typed. For instance, type the following without pressing (END LINE):

```
COS(3)*4
```

Suppose you really wanted the cosine of 2.3 radians not 3 radians. Move the cursor back under the 3 using the (←) key, press (INS RPL), and type 2. . Now the display should show:

```
COS(2.3)*4
```

When you pressed (INS RPL), another cursor appeared to the left of the original cursor. The double cursor informs you that the computer is in insert mode, and tells you that the next character typed will be inserted between the two cursors.

Like the single cursor, the double cursor can be positioned anywhere in the display with the arrow keys. But when you press (END LINE) the second cursor will disappear.

After you have inserted the desired characters, press the (INS RPL) key once again to remove the second cursor from the display and return to replace mode.

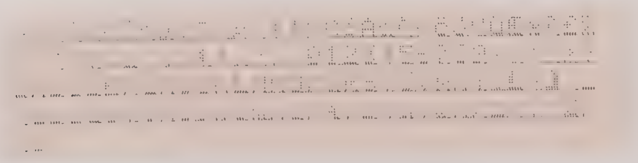
You can insert as many characters as you wish into an expression or a program statement. But make sure that the final expression does not extend beyond three display lines or 95 characters.

If you press (-CHAR) in insert mode, the character above the right cursor will be deleted and the system will return to replace mode automatically.

System Self-Test

Should you feel that the HP-85 is malfunctioning, press the (TEST) key. This causes the system to run through an electronic check of all internal components.

If everything is working properly, the HP-85 system displays and prints the following characters at the end of the test:



These two characters will vary depending on the contents of computer memory.

The graphics display will be cleared, but programs and/or variables in computer memory will remain intact.

If the system is not operating properly, it will display:

```
Error 23 : SELF TEST
```

This tells you that a problem exists in the computer's circuitry; contact the nearest HP dealer or HP sales and service office immediately for system repair.

Resetting the Computer

If the computer becomes inoperative due to a system or input/output malfunction, it may need to be reset. The computer is reset and returned to a ready state by pressing RESET while holding down SHIFT.

Resetting the computer immediately aborts all system activity. The reset operation returns the computer, as well as some peripherals and interfaces, to a ready state. The reset operation is useful when you want to return the system's components to a known configuration before loading or running a program. In other words, RESET sets the trigonometric mode, data pointers, graphics scale and pen, timers, output devices, print all mode, etc., to the same default state as when the system was switched on. If a program is running, any pending or executing input/output operation is terminated and information may be lost.

Refer to the Reset table in appendix C for a list of conditions affected by RESET.

Section 3

Expressions and Keyboard Operations

In this section, we will discuss "expressions" and some of the components of expressions, as well as related keyboard operations. An **expression** is any logical combination of numbers, characters, variables, operators, or functions.

The section's topics include:

- Arithmetic operators.
- Number ranges and number formats.
- Simple numeric and string variables.
- Relational and logical operators.
- Time functions.

The math functions will be discussed in section 4.

So that you'll be familiar with operators, variables, and functions when we use them later in program statements, we'll discuss them in "calculator" mode (from the keyboard, not in programs) now.

Keyboard Arithmetic

You have already become familiar with the numeric keypad. Numeric entry is easy on the HP-85. The HP-85 requires only that you press (END LINE) after the expression is typed, in order to obtain the result.

The arithmetic operations that can be performed on the system are:

- Addition (+)
- Subtraction (−)
- Multiplication (*)
- Division (/)
- Exponentiation (^)
- Integer division (\ or DIV)
- Modulo (MOD)

To perform an arithmetic operation:

1. First key in the expression. (Either the numeric keypad or the typewriter keyboard may be used to type numbers.)
2. Then press (END LINE) to execute the expression.

The result will appear under the line you executed.

For example, multiply 8 by 3:

Press	Display
8 * 3	8*3
END LINE	24

To raise a number to power, such as 8^3:

Press	Display
8 ^ 3	8^3
END LINE	512

You do not need to use parentheses to raise a number to a negative power. For instance, compute 8^{-3}:

Press	Display	
8 ^ -3	8^-3	
END LINE	.001953125	Result.

MOD and DIV

In addition to the usual arithmetic operators, +, −, *, /, and ^, there are two more arithmetic operators that may prove useful to you. These operators are DIV *(integer division)* and MOD *(modulo)*. They are used just like the other five operators.

Integer division (DIV or \) returns the integer portion of the quotient. In other words, normal division takes place, but all digits to the right of the decimal point are truncated (not rounded) so that you only have the whole number result. Integer division can be specified either by keying in DIV or by using the symbol \ for the operator. For example:

Press	Display	
16 DIV 5	16 DIV 5	Key in the expression.
END LINE	3	Then press END LINE.
5 DIV 16	5 DIV 16	
END LINE	0	
5 \ 16	5 \ 16	
END LINE	0	

Given two values A and B, A DIV B = IP(A/B); in other words, the "integer part" of A divided by B.

The MOD (modulo) operator returns the remainder resulting from a division. Like DIV, a normal division occurs, but instead of taking the whole number result as DIV does, MOD takes the remainder and returns it as the result. For instance, when you divide 7 by 3, the division result is 2 with a remainder of 1. MOD would return the 1 as the result of its operation, while DIV would return the 2. For example:

Press	Display	
16 MOD 5 END LINE	16 MOD 5	$16 = 3*5 + 1$
	1	
-8 MOD 3 END LINE	-8 MOD 3	$-8 = -[(2 * 3) + 2]$
	-2	
(-8)MOD 3 END LINE	(-8)MOD 3	$-8 = (-3)*3+1$
	1	

Given two values A and B, `A MOD B=A-B*INT(A/B)`; in other words, A minus B times the greatest integer less than or equal to the quotient of A divided by B. AMOD0 is A, by definition. From the definition, it turns out that $0 \leq AMODB < B$ if $B>0$ and $B<AMODB \leq 0$ if $B<0$.

Despite the fact that `DIV` *can* be spelled out, and `MOD` *must* be spelled out since it has no special symbol, they are still *operators* and are used just like the other five operators are used.

Arithmetic Hierarchy

When an expression has more than one arithmetic operation, the order in which the operations take place depends on the following hierarchy:

`^`	Exponentiation.	Performed first.
`MOD`, `DIV` or `\`, `*`, `/`	Modulo, integer division, multiplication, and division.	↓
`+`, `-`	Addition and subtraction.	Performed last.

When an arithmetic expression contains two or more symbols at the same level in the hierarchy, the order of execution is from left to right.

So an arithmetic expression such as 1+3*2 is equal to 7. The computer performs the multiplication before the addition because of its hierarchy. What if, instead of computing 1+3*2, you really wanted 1+3 and the result times 2? Use parentheses.

Parentheses

The prescribed order of execution can be altered if you use parentheses. Using the example of 1+3 and then multiplying the result times 2, you would type:

`(1+3)*2` (END LINE)	The answer, 8, is returned.

Note that only rounded, `()`, parentheses may be used in numerical operations. The square brackets, `[]`, cannot be used in mathematical calculations.

You may have more than one set of parentheses in an expression, but they must always be "paired up." If you leave out a parenthesis (so that the expression can be said to be "unbalanced"), the HP-85 will return an error message when you press (END LINE)—it won't even try to compute the answer.

When parentheses are used, they take highest priority in the mathematical hierarchy. When parentheses are nested (i.e., when one pair of parentheses is contained inside another pair), like (5*(4−2)), the innermost quantity (4−2) is evaluated first.

Suppose you wish to evaluate the following expression:

$$2 + \frac{3 \times 6}{(7-4)^2}$$

Key it into the computer in one line as follows:

[illegible]

The computer scans an expression from left to right performing the operations of highest priority first. Thus, the above expression would be executed in the following manner:

`2 + 3 * 6 / (7 - 4) ^ 2`	Multiplication.
`2 + 18 / ( 7 - 4) ^ 2`	Evaluate parentheses.
`2 + 18 / 3 ^ 2`	Exponentiation.
`2 + 18 / 9`	Division.
`2 + 2`	Addition.
`4`	Result.

Whenever you are in doubt as to the order of execution for any expression, use parentheses to indicate the order.

Using parentheses for "implied" multiplication is not allowed. So 3(9−5) must appear as `3*(9-5)`. The operator, `*`, must be used explicitly to specify multiplication.

The RESULT Key

The value that is displayed after you press the (END LINE) key to execute a numeric expression is stored in a location called "RESULT." It is obtained for use in other calculations by pressing (SHIFT) (RESLT).

For instance, what if you decided to multiply the result of our last calculation by 3.7:

```
(2+3*6/(7-4)^2)*3.7
```

Press	Display	
(RESLT)*3.7	`4*3.7`	The (RESLT) key immediately displays last result.
(END LINE)	`14.8`	Now 14.8 is the result.

Now suppose you wish to square this result:

Press	Display	
(RESLT) * (RESLT)	`14.8*14.8`	
(END LINE)	`219.04`	Now 219.04 is the result.

PRINT and DISP

The `PRINT` statement and the `DISP` statement are two important program statements. But they can also be used in calculator mode, to have the results of calculations printed, or to output results concurrently. Both of these statements will be discussed further in section 5.

If you wish to display the results of two or more equations simultaneously, use the `DISP` statement and separate your expressions with commas or semicolons. If you use commas, the results will be "spread apart," whereas semicolons will cause the results to be packed together.

Examples:

```
DISP PI*12^2/4,PI*12
 113.097335529
 37.6991118431

DISP 80*43;83*44;86*45;89*46
 3440   3652   3870   4094
```

Execute the statement by pressing (END LINE). Results displayed.

Press (END LINE) to display results.

Or, if you wish to output only the results of your calculations to the printer, use the `PRINT` statement. Press (END LINE) to print results.

Examples:

```
PRINT  222*11,528*8
PRINT  80*43;83*44;86*45;89*46

 2442                4224
 3440   3652   3870   4094
```

Standard Number Format

Your HP-85 has been designed so that for most computations, your results appear in an easy-to-read form, as specified by ANSI*.

All results are calculated with the full precision of the computer. Results are displayed or printed in the following manner unless you specify otherwise in a program statement. (Refer to section 10.)

In standard format:

- All significant digits of a number (maximum of 12 digits) are printed or displayed. For example, if you typed [illegible] it would be output as 99[illegible]19.12.
- Excess zeros to the right of the decimal point are suppressed. For example, `32.100000` would be output as [illegible]
- Leading zeros are truncated. For example, `00223.` is output as `223.`
- Numbers whose absolute values are greater than or equal to 1, but less than 10^{12} are output showing all significant digits and no exponent.
- Numbers between −1 and 1 are also output showing all significant digits and no exponent if they can be represented precisely in 12 or fewer digits to the right of the decimal point.
- All other numbers are expressed in scientific notation.

Let's look at a few examples of standard format. In the following table, if you key in the number in the left column and press (END LINE), that number will be displayed in the format shown in the right column.

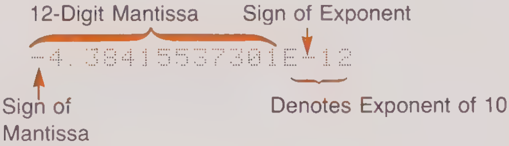
---

* American National Standards Institute.

Number	Standard Format
15.000	15
00.23500	.235
-.0547^9	-4.38415537301E-12
000987.5	987.5
10000^6	1.E24
.01E4	100
120E-4	.012

Scientific Notation

In the right-hand column above, you see two numbers expressed in scientific notation. When you execute an expression in which the result is too large or too small to be displayed fully in 12 digits, the number is displayed with a single digit to the left of the decimal point, followed by up to 11 digits to the right of the decimal point, followed by the letter E and an exponent of 10.



For example:

Press	Display	
60000*90000000	60000*90000000	
END LINE	5.4E12	Result, 5.4 × 10^{12}.
.00006*.00000009	.00006*.00000009	
END LINE	5.4E-12	Result, 5.4 × 10^{-12}.

Keying In Exponents of Ten

You can key in numbers multiplied by powers of 10 (like the last two examples in the table above), by typing the number, then E, followed by an exponent of 10. For example, to key in 15.6 trillion (15.6 × 10^{12}) and multiply it by 25:

Press	Display	
15.6E12*25	15.6E12*25	
END LINE	3.9E14	Result.

To key in negative exponents of 10, type the number, type E, and then type the negative exponent. For example, type Planck's constant (h)—roughly, 6.625 × 10^{-27} erg seconds—and multiply by −50.

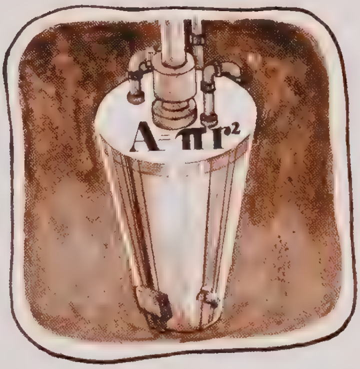
Press	Display	
6.625E-27*-50	6.625E-27*-50	
END LINE	-3.3125E-25	Erg seconds.

Range of Numbers

The range of values which can be entered or stored is $-9.99999999999 \times 10^{499}$ through -1×10^{-499}, 0, and 1×10^{-499} through $9.99999999999 \times 10^{499}$.

Variables

Algebraic formulas usually contain names that represent assigned values. These names are known as variables and, with the HP-85, specify a location in memory where a value is stored. For instance, the formula for the area of a circle, $a=\pi r^2$, contains two variables, a and r. To use the formula, you assign a value to r (radius) to solve for a.



Types

With the HP-85 you can specify either numeric variables or "character string" variables. Characters strings, or "strings" for short, can be composed of any valid characters and can be of any length—from zero characters to a maximum limited only by available memory. But since numeric data is more often used, we will discuss numeric variables first, then touch briefly on string variables. We'll continue our discussion on variables in section 8.

There are three types of numeric variables allowed by the HP-85.

- `REAL` numbers are stored with the full precision of the computer. `REAL` numbers are represented internally with 12 digits and a three-digit exponent in the range of –499 through 499; in other words, a 12-digit number in the range $-9.99999999999 \times 10^{499}$ through $-1.00000000000 \times 10^{-499}$, 0, and $1.00000000000 \times 10^{-499}$ through $9.99999999999 \times 10^{499}$.
- `SHORT` numbers are represented internally with five digits and a two-digit exponent in the range –99 through 99; in other words, a five-digit number in the range of -9.9999×10^{99} through -1.0000×10^{-99}, 0, and 1.0000×10^{-99} through 9.9999×10^{99}.
- `INTEGER` numbers are stored with five digits, with no digits following the decimal point. The range of integers is –99999 through 99999.

All numbers are full precision (real) unless you specify otherwise. But you can conserve computer memory if you designate `SHORT` or `INTEGER` numbers; refer to page 141.

Forms

There are two forms that a variable may have:

- Simple.
- Array (subscripted).

With simple variables, you assign a numeric value or expression to a name. Arrays are convenient for handling large groups of data within a program.

Simple variables can be assigned values in either calculator mode or within a program.

Calculator mode variables are temporary—they are cleared from memory whenever you run a program or press SCRATCH or RESET. Use them when you want to calculate immediate results from the keyboard. Otherwise, use variables in programs, where you can use them over and over again. The following statements about variable names and assignments apply to both calculator mode variables and program variables.

Simple Variables

On the HP-85 you can use the following for simple variable names:

- Any letter from A through Z. (Small letters can be used, but they are interpreted as if they were capital letters.)
- Any letter immediately followed by a digit from 0 through 9.

For instance, acceptable simple variables names are: A1, c, F0, j, j5, r2, x, Y.

Note: Lower case variable names are turned into upper case letters by the system (e.g., a1 is the same as A1).

In all, 286 simple variables can be named.

Variables are assigned values using an equal sign to create an assignment statement. For example, to assign 15 to A and 2*25 to X3:

Press	**Display**
A = 15 END LINE	A = 15
X3 = 2*25 END LINE	X3 = 2*25

In the assignment statement, the variable name appears first, followed by the equal sign. The value or numeric expression assigned to the variable appears to the right of the equals sign.

Now that some variables have assigned values, they can be used in place of numbers in math calculations:

Press	**Display**	
A/X3 END LINE	A/X3	
	.3	Result of 15/50.
A∧2 END LINE	A^2	
	225	Result of 15^2.
X3 * 3 END LINE	X3*3	
	150	Result of 50 * 3.

Variables can be reassigned values. For instance to change the value of A to 16 you could execute either A = A+1 or A = 16.

To recall the value of any assigned variable, simply type the variable name and press END LINE.

Press	**Display**	
A END LINE	`A`	
	`15`	Value of A.
X3 END LINE	`X3`	
	`50`	Value of X3.

You can assign the same value to more than one variable in the same line by using commas to separate the variables. For example, assign the variables A, B, C, and D the value of 100.

Press	**Display**	
A, B, C, D = 100 END LINE	`A,B,C,D=100`	
A END LINE	`A`	Verify that all variables have been assigned the value 100.
	`100`	
B END LINE	`B`	
	`100`	
C END LINE	`C`	
	`100`	
D END LINE	`D`	
	`100`	

You can use one more type of numeric variable on the HP-85—an array variable. We'll discuss arrays in section 8.

String Variables

A character string is a series of characters like `**HELLO!**` that can be given a string variable name. The length of the string refers to the number of characters assigned to the string. A string variable can be any length (limited only by available memory).

You can use string variables without dimensioning them (allocating memory to them) if they contain less than 18 characters. If they are longer than 18 characters, you must use a `DIM` or `COM` statement to declare the length (page 121).

String variables are assigned names in the same way that numeric variables are assigned names, but the string variable name must be followed by a dollar sign (`$`).

For example, acceptable string variable names are: `A1$`, `C$`, `F0$`, `j$`, `_5$`, `r2$`, `X$`, `Y$`.

In all, 286 string variables can be named. (Remember, the system interprets small letters in variable names as if they were capital letters; thus, you could reference the same string variable `A1$` with `a1$`.)

To assign HELLO to A$, and GOODBYE to B$:

Press	Display	
A$=“HELLO” END LINE	`A$="HELLO"`	The string must be enclosed with quotation marks.
B$=“GOODBYE” END LINE	`B$="GOODBYE"`	
DISP A$,B$ END LINE	`DISP A$,B$`	
	`HELLO                GOODBYE`	
DISP A$;“–”;B$ END LINE	`DISP A$;"-";B$`	
	`HELLO-GOODBYE`	

String Concatenation

“Concatenation” is the one operation allowed in string expressions. This operation causes one string to be tacked onto the end of another. The symbol used for string concatenation is the ampersand (&). To join two strings together, it is necessary only to interpose the ampersand.

For example, assign the following string variables the characters shown:

```
A$="BUTTER"
B$="DRAGON"
C$="HOUSE"
D$="FLY"
E$=" "
```

Press END LINE after each assignment statement.

The string variable E$ contains one space.

Now execute these statements:

```
DISP A$ & D$ END LINE
BUTTERFLY
```

Displays the two strings joined together.

```
F$=B$ & D$ END LINE
F$ END LINE
DRAGONFLY
```

Joins strings B$ and D$ to make F$.

```
DISP F$ & E$ & C$ & D$ END LINE
DRAGONFLY HOUSEFLY
```

Since concatenation makes a string longer than its parts, be sure that the final string in a string variable assignment is less than or equal to 18 characters in length. (Or, if the string has been dimensioned using `DIM` or `COM` statement in a program, the final string must not exceed the length that you have designated.)

The Null String

The null string is a string that contains no characters or blanks, like:

```
H$=""
```

We define the null string here because it is referred to often.

Logical Evaluation

In logical evaluation, expressions can be compared by using relational operators and/or logical operators. An expression can be a constant (like `7.2`), a variable (like `B`), or an arithmetic expression (like `7.2*SQR(6)`). If the comparison is 'true', the value '1' is returned; if the comparison is 'false', the value '0' is returned.

Relational Operators

Relational operators are used to determine the value relationship between two expressions.

Symbol	Meaning
=	Equal to.
<	Less than.
>	Greater than.
<=	Less than or equal to.
>=	Greater than or equal to.
<> or #	Not equal to (either form is acceptable).

The < symbol corresponds to the shifted (<;) key, > corresponds to the shifted (>.) key, and # corresponds to the shifted (#3) key.

Be careful to note that the equal sign is used in both variable assignment statements and in relational operations. This distinction only becomes important at the beginning of an expression that could be interpreted either way; in which case, the system will always assume that the expression is a variable assignment. To specify a relational operation, place parentheses around the equality relational operation or place the value to the left of the equal sign and the variable it is being compared with to the right of the equal sign.

Examples:

`A=3`	Assigns A the value of 3.
`(A=3)` `3=A`	Both expressions perform the equality relational operation and compare the value of A with 3. They return values of 0 or 1 depending on whether A has a value of 3.
`A,B=3`	Assigns A and B the value of 3.
`A=B=3` `A=(B=3)`	Both statements assign the value 0 or 1 to A, depending on whether B does or does not equal 3. You don't need to use parentheses since the variable name is to the left and its value (the result of the expression B=3) is to the right of the equal sign.

Let's look at some examples using relational operations. First let's assign values to the variables A,B,C, and D.

```
A = 1
B = 2
C,D=3
```

Press (END LINE) after each line to assign the variable(s) the specified value.

Now execute the following operations:

```
A < B                    1 < 2
 1                       True.
B < A                    2 < 1
 0                       False.
B # C                    2 # 3
 1                       True.
C # D                    3 # 3
 0                       False.
3 = C                    3 = 3
 1                       True.
4 = A                    4 = 1
 0                       False.
A = 4                    Assigned 4 to A.
```

As you can see, the last statement did not assign a value of `1` (true) or `0` (false) to the expression because `A = 4` is an assignment statement; so A is assigned the value of 4. To determine the value relationship between the value of A and 4, type `4 = A`, as shown above, or use the parentheses around this expression: (`A=4`).

Strings and string variables can also be compared using the relational operators. Each character in the string is represented by a standard decimal code, as shown in appendix C. When two string characters are compared, the lesser of the two characters is the one whose decimal code is smaller. For example, 3 (decimal code 51) is smaller than B (decimal code 66).

Strings are compared, character by character, from left to right until a difference is found. If one string ends before a difference is found, the shorter string is considered the lesser.

Relational operators are valuable when they are used in `IF` ... `THEN` statements as described in section 7.

Advanced Programming Note: Relational comparisons can be quite complex. Suppose the following statements are used in a program:

```
X=69*(A=3)+287*(B=83)              Assigns 69 to X when A=3 and adds 287
                                   when B=83, otherwise adds nothing.
G1=4*(A$="A")+3*(A$="B")+2*(A$="   Assigns 4 for an "A", 3 for a "B", 2 for a
C")+(A$=D$)                        "C" and 1 when A$=D$.
L=L+(LEN(A$)>9)                    Adds 1 to L when A$ is longer than nine
                                   characters.
```

Logical Operators

The logical operators, often called 'Boolean' operators, are `AND`, `OR` (inclusive or), `EXOR` (exclusive or), and `NOT`. A value of zero is considered false. Any other value is considered true. The result of a logical operation is either 0 or 1.

- `AND` checks two expressions. If both expressions are true, (that is, both non-zero), the result is true (`1`). If one or both of the expressions is false (`0`), the result is false (`0`).

A	B	A AND B
T	T	T
T	F	F
F	T	F
F	F	F

- `OR` checks two expressions. If one or both of the expressions is true, the result is true (`1`). If neither expression is true, the result is false (`0`).

A	B	A OR B
T	T	T
T	F	T
F	T	T
F	F	F

- `EXOR` (exclusive or) compares two expressions. If only one of the expressions is true, the result is true (`1`). If both are true or both are false, the result is false (`0`).

A	B	A EXOR B
T	T	F
T	F	T
F	T	T
F	F	F

- `NOT` returns the opposite of the logical value of an expression. If the expression is true (non-zero), the result is false (`0`). If the expression is false (`0`), the result is true (`1`).

A	NOT A
T	F
F	T

We have used A and B in the truth tables to denote numeric expressions. The expressions used with logical operators can be either relational or non-relational. In the order of execution of expressions, `NOT` has higher priority than the relational operators and the relational operators have higher priority than `AND`, `EXOR`, and `OR`. If you are in doubt, use parentheses.

Here are some examples; first let's assign values to the variables A,B,C, and D.

```
A = 0
B = 2
C,D = 4
```

Press (END LINE) after each line to assign the variables the specified values.

Now execute these logical expressions:

```
A<B AND C=D
 1
```

Since both relational expressions `A<B` and `C=D`, are true, the result is true.

```
A AND C=D
 0
```

The expression, `A`, is false since its arithmetic value equals zero. The expression, `C=D`, is true. But since `AND` requires that both expressions be true to return a true result, the result is false.

```
A OR B
 1
```

The arithmetic value of `A` is zero (false) while the arithmetic value of `B` is two (true). Since at least one of the expressions is true, the whole expression is true.

```
A OR C-D
 0
```

Both arithmetic expressions have a value of 0 (false).

```
NOT A
 1
```

Since `A` is false. `NOT A` is true.

```
A EXOR B=2
 1
```

Since `A` is false and `B=2` is true, the result is true.

Advanced Programming Note: The results returned from logical or relational operations, either 0 or 1, can be used in calculations. Using the variables, A, B, C, and D again, let's evaluate S in the equation shown below:

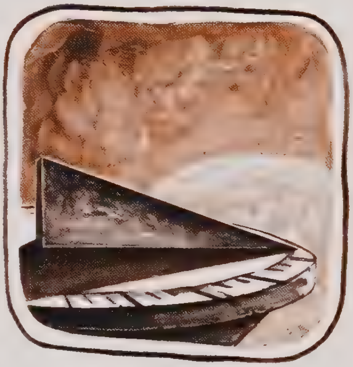
```
S = ((B<C) + (NOT D=A))*12 END LINE
S END LINE
 24
```

The result of the true relation (`B<C`) is first added to the result of the true relation (`NOT D=A`). In other words 1 + 1 = 2. This result is then multiplied by 12 for a product of 24.

Here's a truth table summarizing logical operations:

A	B	A AND B	A OR B	A EXOR B	NOT A	NOT B
T	T	1	1	0	0	0
T	F	0	1	1	0	1
F	T	0	1	1	1	0
F	F	0	0	0	1	1

The Time Functions



Often it is desirable to document programs, computations, and test runs with the current time and date of execution. With the HP-85 you can set the time and date and then recall the current time whenever you wish. You can even use the time functions in calculations.

As soon as you turn the power on, the system timer is set to 0 and begins to count the time in milliseconds. After it counts 86,400 seconds (24 hours), the system timer increments the date by 1, and then starts to count milliseconds from 0 again.

You can specify the starting time and date for the system counter with the `SETTIME` statement as follows:

`SETTIME` seconds since midnight , day of the year

Although the time must be set in seconds to count properly, the date can be specified any way you want—as long as you remember that the date is an integer number that is incremented by 1 at midnight (assuming the system timer has been set properly).

For example, you can set the timer at 8 a.m., March 16, 1980 as follows:

```
SETTIME 28800,80076
```

28800 seconds since midnight, 76th day of 1980; date in form yyddd.

Eight o'clock in the morning is 8 hours × 60 minutes/hour × 60 seconds/minute = 28800 seconds since midnight. And March 16, 1980 is the 76th day of 1980.

Now the timer will increment 28800.000 by 1 every millisecond until the time is 86400 (midnight). Then it will add 1 to 80076 and start counting seconds from 0 again.

Since the date is just an integer number that is incremented by 1 every 24 hours, you can enter the date in any form you wish, as long as the number is between 1 and 99999. The time parameter can be a numeric expression with a value of 0 through 86400. If the timer is set to 86400, midnight, the system immediately increments the date and begins counting milliseconds from 0 again.

For instance, you could have set the date and time for 8 a.m., March 16th as follows:

`SETTIME 8*60*60,316`	Date in form mdd.

Notice that you can use a numeric expression to set the time. We used the number 316 to specify the 3rd month and the 16th day, but remember, the system interprets 316 as just a number to be incremented by 1 at midnight.

The `TIME` function recalls the current time in seconds since midnight, assuming the time has been set properly, or the number of seconds since power on if the time was not set with `SETTIME`. To recall the time, type:

`TIME` END LINE

The `DATE` function recalls the current date in the same format that you specified, or it recalls 0 if you had not set the date with `SETTIME`. To recall the date, type:

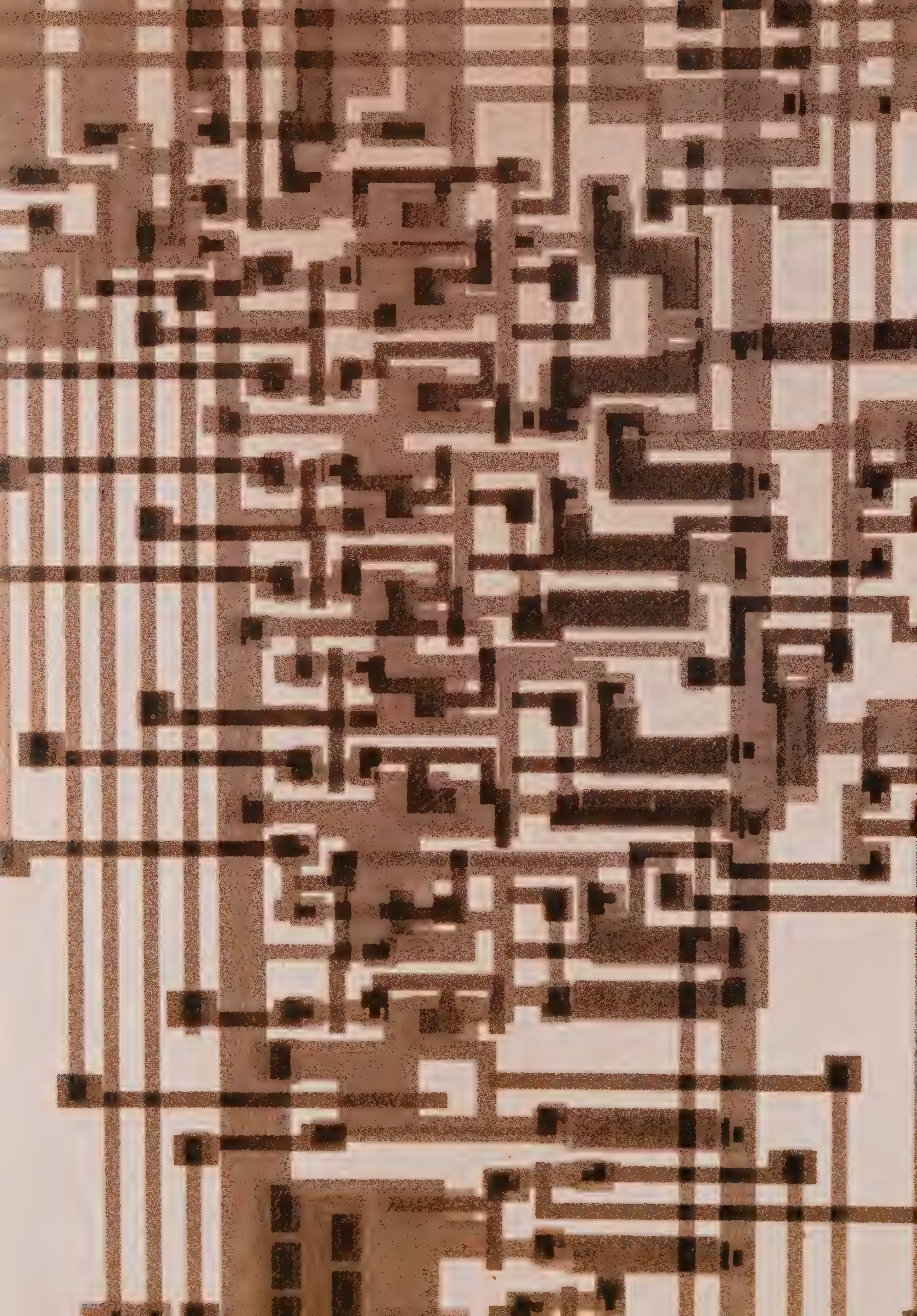
`DATE` END LINE

All values for `SETTIME`, `TIME`, and `DATE`, are lost when you turn the power off. `TIME` begins counting from 0 each time the power is turned on.

The `TIME` and `DATE` functions are programmable and can be used in numerical expressions.

For instance, to recall the time in hours, execute:

`TIME/3600` END LINE	Time in seconds divided by (60 minutes/hour × 60 seconds/minute). Result gives time in decimal hours.



Section 4

Mathematics Functions and Statements

Many predefined functions are available to you through BASIC programming language on the HP-85. But you don't need to write a program in order to use them. Each function operates the same way, regardless of whether you execute the function straight from the keyboard or use it as part of a program statement.

In this section:

- Each built-in math function is explained as it is used manually, in its simplest form.
- The math functions are placed in the total math hierarchy.
- Math errors are discussed in conjunction with the `DEFAULT ON` statement.

A **function** is a prescription for doing something with a given value, or set of values, that yields a single output. The values that are acted upon by a function are called the "arguments" or, sometimes, the "parameters." An argument is often just a single number, but it may be a mathematical expression containing variables or other functions.

Most of the functions on the HP-85 require only one argument, but there are several that require two, and a few that require none.

To use any of the functions in "calculator" mode:

1. Type the function name.
2. Then type the argument, if the function requires one, enclosed within parentheses. If the function requires two arguments, separate them with a comma.
3. Press (END LINE) to compute the result.

Appendix D lists all of the functions available to you with BASIC on the HP-85.

Number Alteration

There are several functions that allow you to alter numbers on the HP-85. These functions are: `ABS`, `INT` or `FLOOR`, `CEIL`, `FP`, and `IP`. The table below lists the function name and argument along with the meaning of the function. The argument X may be a number (like `2.75`), a variable (like `A`), or a numeric expression (like `3*SQR(A)`).

Function and Argument	Meaning
`ABS(X)`	Absolute value of X.
`IP(X)`	Integer part of X.
`FP(X)`	Fractional part of X.
`INT(X)`	Greatest integer less than or equal to X.
`FLOOR(X)`	Greatest integer less than or equal to X. (Same as `INT(X)`; relates to `CEIL`.)
`CEIL(X)`	Smallest integer greater than or equal to X.

Note: `IP` and `INT` differ only with negative numbers.

Absolute Values

Some calculations require the absolute value, or magnitude, of a number. To obtain the absolute value of any expression, simply type `ABS(`expression`)`, where the expression may be a constant, a variable, or an arithmetic expression. Then press [END LINE]. The result will be displayed below the line you type.

Examples:

```
ABS(-235)
 235
```

Press [END LINE] to display the result $|-235|$.

```
ABS(2.7)
 2.7
```

$|+2.7|$

```
ABS(4-7/1.5)
 .666666666667
```

$|4-7/1.5|$

Integer Part of a Number

To extract and display the integer part of a number, type `IP`, followed by the argument enclosed within parentheses. Then press [END LINE].

Examples:

```
IP(123.456)
 123
```

Press [END LINE] and the integer part of the number is displayed.

```
IP(-4.56)
-4
```

Integer part of −4.56.

```
IP(1.748)
 1
```

Integer part of 1.748.

When the `IP` function is executed, the fractional part of the number is lost.

Fractional Part of a Number

To extract and display only the fractional part of a number, type `FP`, followed by the argument enclosed within parentheses. Then press [END LINE].

Examples:

```
FP(123.456)
 .456
```

When you press [END LINE], only the fractional part of the number is displayed.

```
FP(-4.56)
-.56
```

Fractional part of −4.56.

```
FP(1.748)
 .748
```

Fractional part of 1.748.

When the `FP` function is executed, the integer part of the number is lost.

Greatest Integer Function

To display the greatest integer less than or equal to a number, type `INT` or `FLOOR`, followed by the number or expression enclosed within parentheses. The greatest integer function returns the largest integer that is less than or equal to the evaluated expression.

Examples:

```
INT(123.456)
 123
```

123 is the greatest integer <= 123.456.

```
FLOOR(123.456)
 123
```

FLOOR performs the same function as INT.

```
INT(-6.257)
-7
```

−7 is the greatest integer <= −6.257.

```
INT(-1.748)
-2
```

−2 is the greatest integer <= −1.748.

Note the difference between the IP *(integer part)* function and the INT or FLOOR *(greatest integer)* function. In the above examples, `IP(-6.257)` yields `-6`, while `INT(-6.257)` yields `-7`.

Smallest Integer Function

To display the smallest integer greater than or equal to a number, type CEIL *(ceiling)*. Then type the number or expression, enclosed within parentheses, and press END LINE.

Examples:

```
CEIL(123.456)
 124
```

124 is the smallest integer >= 123.456.

```
CEIL(-6.257)
-6
```

−6 is the smallest integer >= 6.257.

```
CEIL(-1.748)
-1
```

−1 is the smallest integer >= −1.748.

General Math Functions

Several of the following functions do not require an argument. For instance, PI always returns the 12-digit approximation of π. A few of the functions below require two arguments; for example, given two values, MAX returns the larger of the two values. The arguments, denoted by X and Y, may be numbers, numeric variables, functions, or numeric expressions.

Function and Argument	Meaning
SQR(*X*)	Positive square root of *X*.
SGN(*X*)	Sign of *X*; yields −1 if $X < 0$, 0 if $X = 0$, and +1 if $X > 0$.
MAX(*X*,*Y*)	Maximum; if $X > Y$ returns *X*, otherwise returns *Y*.
MIN(*X*,*Y*)	Minimum; if $X < Y$ returns *X*, otherwise returns *Y*.
RMD(*X*,*Y*)	Remainder of *X* divided by *Y*: $X-Y$*IP(X/Y).
PI *no argument*	12-digit approximation of π; 3.14159265359.
INF *no argument*	Machine infinity (+9.99999999999E499).
EPS *no argument*	Epsilon; smallest positive machine number (1E-499).
RND *no argument*	Random number; generates next number in a sequence of numbers greater than or equal to zero and less than one.

Square Root Function

To calculate the square root of a number, use the `SQR` function. The square root function returns the square root of a nonnegative expression.

Examples:

```
SQR(88)
 9.38083151965

SQR(16.1)
 4.01248052955
```

When you press END LINE, the square root of the number is displayed.

Sign of a Number

The sign function returns a 1 if the expression is positive, 0 if it is 0, and −1 if it is negative. To use the sign function, type `SGN`, followed by the argument enclosed within parentheses. Then press END LINE.

Examples:

```
SGN(-5)
-1

SGN(0)
 0

SGN(4.3)
 1
```

Sign of a negative number is −1.

Sign of zero is 0.

Sign of a positive number is 1.

Maximum and Minimum

You'll find the `MAX` and `MIN` functions very useful in BASIC programs. Given two values, `MAX` returns the larger of the values and `MIN` returns the smaller. Both `MAX` and `MIN` require two arguments enclosed within parentheses, following the function name. And these two arguments must be separated by a comma.

Examples:

```
MAX(4.5,4.76)
 4.76

MIN(4.5,4.76)
 4.5

MAX(-1,-5)
-1

MIN(-1,-IP(SQR(5)))
-2
```

Note that the arguments must be separated by a comma. The arguments may be numbers, simple variables—if the variables are defined—or arithmetic expressions (including functions).

The Remainder Function

Given two values, the remainder function, `RMD`, divides the first value by the second and displays the remainder. `RMD(X,Y)= X-Y*IP(X/Y)`.

Examples:

```
RMD(5,2)
 1
```

2 goes into 5 twice, with a remainder of 1.

```
RMD(17.35,4.26)
 .31
```

4 times 4.26 plus remainder .31 is equal to 17.35.

Comparing definitions, you can see that the `RMD` function and the `MOD` operator (page 44) are very similar. In fact, they both yield the same results when the arguments X and Y have the same sign. But they can give different results when X and Y are of opposite signs.

Whether you use `RMD` or `MOD` depends on the particular application you choose.

Example: Resolve −726° to lie between −360° and +360° by ignoring multiples of 360°. Using the `RMD` function, given any θ in degrees:

$$\theta_{\text{new}} = \text{RMD}\,(\theta, 360) \text{ where } -360° < \theta_{\text{new}} < 360°$$

```
RMD(-726,360)
-6
```

With the `RMD` function, θ_{new} is equal to −6°.

Now use the `MOD` operator to resolve −726° to lie between 0° and 360° by ignoring multiples of 360°. Given any θ in degrees:

$$\theta_{\text{new}} = \theta\text{MOD}360 \text{ where } 0° < \theta_{\text{new}} < 360°$$

```
(-726)MOD360
 354
```

$\theta_{\text{new}} = 354°$

Using PI

The value of π approximated to 12 places (3.14159265359) is provided as a fixed constant in BASIC programming language. Merely type `PI` whenever you need it in a calculation. For example, to calculate 3π, type:

```
3 * PI
 9.42477796077
```

When you press END LINE, the result is displayed.

Example: Calculate the surface area of Callisto, one of Jupiter's 12 moons, using the formula $A = \pi d^2$. Callisto has a diameter (d) of 3100 miles.

```
PI * 3100 ^ 2
 30190705.401
```

Area of Callisto in square miles.

Note that you don't have to include parentheses around `3100 ^ 2` because exponentiation is performed before multiplication.

Epsilon and Infinity

Two functions that prove useful in programs are `EPS` and `INF`. Both functions simply recall a constant; `EPS` recalls the smallest positive machine number and `INF` recalls the largest machine number. They are useful in comparisons when you want to use either a very small or a very large number, saving you the time of keying in the numbers yourself. (Recall how we used `EPS` in the Averaging Program in section 1.)

Examples:

```
EPS
 1.E-499

INF
 9.99999999999E499

MAX(458, INF)
 9.99999999999E499

MIN(32, EPS)
 1.E-499
```

Smallest positive number that can be output: 1 × 10^{-499}.

Largest number that can be output: 9.99999999999 × 10^{499}.

Random Numbers

Random numbers are extremely useful in statistical sampling theory—anytime you want a sequence of events or numbers that appear in an unpredictable order. The random number function, `RND`, returns a pseudorandom number greater than or equal to 0 and less than 1, each time it is executed.

Example:

```
RND
 .529199358633

RND
 4.35821814444E-2

RND
 .294922982088
```

A random number between 0 and 1 is displayed each time RND is executed.

Whenever you turn the power on or press RESET, the same sequence of random numbers is generated. The reason for this is that the random number function uses the same 'seed' (i.e., the number upon which the sequencing is based) each time it is reset.

But, by using the `RANDOMIZE` statement, you can ''scramble'' the seed and thus, generate new sequences of random numbers. Or you can control the sequences of random numbers by specifying your own ''seed.''

To see how this works, type:

```
RANDOMIZE
```

When you press END LINE, the HP-85 defines a new seed for the random number generator, based on the internal timing system. Now, execute the `RND` function several times, until it becomes evident that you have generated a new sequence of random numbers.

Each time you use the `RANDOMIZE` statement in this way, a new "seed" is defined, yielding a new sequence of random numbers.

You can control the sequence of numbers by specifying the "seed" with the `RANDOMIZE` statement. This enables you to regenerate the same sequence of numbers whenever you wish.

Example: Using the seed .423, generate the first three numbers of the random number sequence.

```
RANDOMIZE .423
RND
 .543851130928
RND
 .50809747097
RND
 .484429755623
```

Execute the `RANDOMIZE` statement.

1st random number in the sequence.

2nd random number in the sequence.

3rd random number in the sequence.

Whenever you wish to use the same sequence of random numbers, use the same seed. To obtain a good seed, use any non-zero number within the range of the HP-85—the system will automatically convert the number to a seed between 0 and 1. A seed of zero will generate a constant sequence of zeros.

For any non-zero seed in the given range, 5×10^{13} values are generated before the sequence repeats. (The `RANDOMIZE` statement will always generate a non-zero seed if no parameter is specified.)

You are not limited to random numbers between numbers 0 and 1. In general, you can generate random integers from a through b using the formula $\mathrm{IP}((b + 1 - a) * \mathrm{RND} + a)$. For instance, generate a random sequence of integers between 0 and 99, inclusive.

You could use an expression like the following:

```
IP(100*RND)
 5
IP(100*RND)
 54
IP(100*RND)
 6
```

The integer part of a random number times 100. (These are the fourth, fifth, and sixth random numbers generated in the sequence based on the seed above.)

Generally speaking, good statistical properties can be expected because the random number generator has been designed to pass an important test known as the spectral test.* Of course the statistics will vary somewhat from sequence to sequence depending on the starting seed since less than the full period will be used by you. But it should normally be quite good if a "statistically significant" sample size is considered.

Logarithmic Functions

The HP-85 computes both natural and common logarithms as well as their inverse functions. The logarithmic functions are:

Function and Argument	Meaning
`LOG` *X*	$\log_e X$; natural logarithm of a positive X to the base e (2.71828182846 to 12 place accuracy).
`EXP` *X*	e^X; natural antilogarithm. Raises e (2.71828182846) to the power X.
`LGT` *X*	$\log_{10} X$; common logarithm of a positive X to the base 10.

---

* Donald E. Knuth, *The Art of Computer Programming* (Massachusetts, 1969), V.2., §3.4

Of course, the common antilog (10^x) may be executed easily from the keyboard (`10^X`).

Example: What is the value of $\log_2 53$?

You can easily convert the logarithmic base using the following formula:

$$\log_a x = \frac{\ln x}{\ln a} = \frac{\log_e x}{\log_e a}$$

So, to find the logarithm, base 2, of 53, simply execute:

```
LOG(53)/LOG(2)
 5.72792045456
```

$\log_2 53$.

Trigonometric Functions and Statements

Trigonometric Modes

When you are using trigonometric functions, angles can be assumed by the HP-85 to be in decimal degrees, radians, or grads. Unless you specify otherwise with one of the trigonometric mode statements, the HP-85 assumes that all angles are in radians. When you select a trigonometric mode, the HP-85 remains in that mode until you change it or the computer is turned off.

To select degrees mode, execute:

```
DEG
```

There are 360 degrees in a circle.

To select grads mode, execute:

```
GRAD
```

There are 400 grads in a circle.

To reset radians mode, press RESET or execute:

```
RAD
```

There are 2π radians in a circle.

$$360 \text{ degrees} = 400 \text{ grads} = 2\pi \text{ radians}$$

Trigonometric Functions

There are 12 programmable trigonometric functions provided by the HP-85, including inverses of several of the functions and conversion functions.

Function and Argument	Meaning
`SIN(X)`	Sine of X.
`ASN(X)`	Arcsine of X; $-1 \le X \le 1$. In 1st or 4th quadrant.
`COS(X)`	Cosine of X.
`ACS(X)`	Arccosine of X; $-1 \le X \le 1$. In 1st or 2nd quadrant.
`TAN(X)`	Tangent of X.
`ATN(X)`	Arctangent of X; in 1st or 4th quadrant.
`CSC(X)`	Cosecant of X.
`SEC(X)`	Secant of X.
`COT(X)`	Cotangent of X.
`ATN2(Y,X)`	Arctangent of Y/X, in proper quadrant; useful in polar/rectangular coordinate conversions.
Conversions:	
`DTR(X)`	Degrees to radians conversion.
`RTD(X)`	Radians to degrees conversion.

All trigonometric functions have one argument, except the `ATN2` function, so to use them simply type the function name and then type the numeric expression, enclosed within parentheses.

Example: Find the cosine of 45 degrees.

```
DEG
COS(45)
 .707106781187
```

Sets the HP-85 to degrees mode.

Result.

Example: Find the sine of 2/3π radians.

```
RAD
SIN(2/3 * PI)
 .866025403786
```

Sets computer to radians mode.

Result.

Degrees/Radians Conversions

The `DTR` *(degrees to radians)* and `RTD` *(radians to degrees)* functions are used to convert angles between degrees and radians. To convert an angle specified in degrees to radians, type `DTR` followed by the angle within parentheses. For example, to change 45 degrees to radians:

```
DTR(45)
 .785398163397
```

Radians.

To convert the angle specified in radians to decimal degrees, type `RTD` followed by the angle within parentheses. Convert 4 radians to decimal degrees:

```
RTD(4)
 229.183118052
```

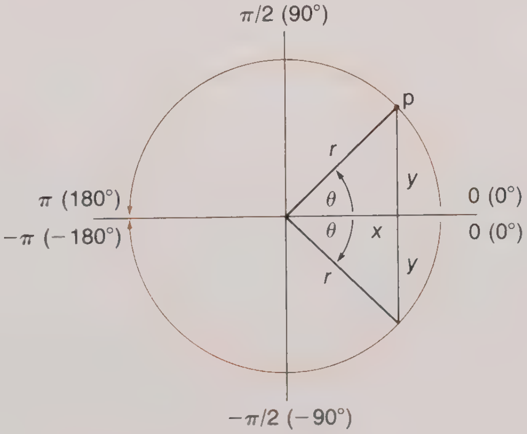
Decimal degrees.

Polar/Rectangular Coordinate Conversions

The `ATN2` *(arctangent of x,y coordinate position)* function can be used for polar/rectangular coordinate conversions. Angle θ is assumed in decimal degrees, radians, or grads, depending upon the trigonometric mode first selected by `DEG`, `RAD`, or `GRAD`.

A point P can be represented in two ways: by the rectangular coordinate position (x,y) or by the polar coordinate position (r,θ).

In the HP-85, the `ATN2` function produces an angle θ represented in the following manner:



To convert from rectangular (x,y) coordinates to polar (r,θ) coordinates (magnitude and angle, respectively), use the following equations:

$$r = \sqrt{x^2 + y^2}$$

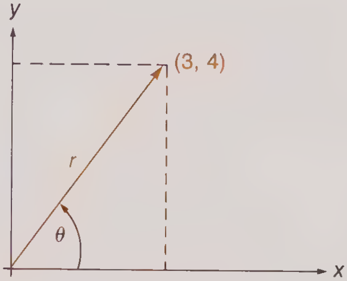
$$\theta = \mathrm{ATN2}(y,x) \qquad \text{where } -\pi < \theta \leq \pi.$$

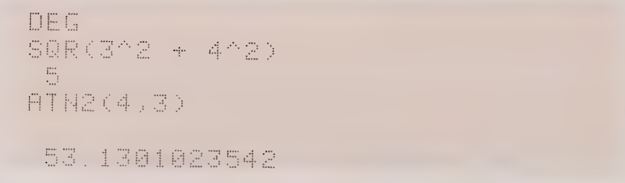
To convert from polar (r,θ) coordinates to rectangular (x,y) coordinates, use the following geometric properties:

$$x = r\cos\theta$$

$$y = r\sin\theta$$

Example: Convert rectangular coordinates (3,4) to polar form with the angle expressed in decimal degrees.



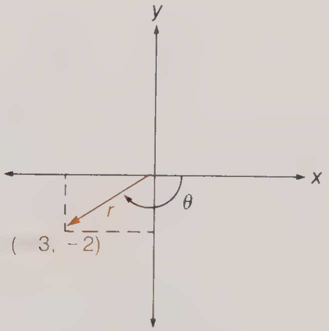


```
DEG
SQR(3^2 + 4^2)
 5
ATN2(4,3)

  53.1301023542
```

Degrees mode selected.
$r = \sqrt{x^2 + y^2}$.
Magnitude r.
θ = ATN2(y,x); notice that we specify the y-coordinate value first.
Angle θ in decimal degrees.

The ATN2 function is also used to find the arctangent of an expression in the *proper* quadrant. The ATN function returns the principal value of the arctangent of an expression, in other words, the value in the first or fourth quadrant.

Example: Find the angle in the third quadrant whose tangent is 2/3. Express the angle in radians.



```
RAD
ATN2(-2,-3)


-2.55359005004
```

Set radians mode.
θ = ATN2(y,x), where x and y are any rectangular coordinates in the third quadrant with a tangent of 2/3.
Angle θ in radians.

(Note that `ATN(-2/-3)` would return the arctangent of 2/3 evaluated in the *first* quadrant: `.588002603548`.)

Total Math Hierarchy

Notice that functions are performed immediately after parentheses are evaluated in the order of execution for all mathematical operations.

Highest priority `( )` parentheses
Functions
`^` (exponentiation)
`NOT`
`*`, `/`, `MOD`, `\` or `DIV`
`+`, `-`
All relational operators (`=`, `>`, `<`, `>=`, `<=`, `<>` or `#`)
`AND`
Lowest priority `OR`, `EXOR`

Recovering From Math Errors

Many math errors occur due to an improper argument or overflow. Such an error would normally halt the execution of a running program. The HP-85 provides default values for out-of-range results that occur using the following math functions, thus overriding the error condition and preventing the error from halting program execution. The system will alert you to the error by displaying a warning message and, if the result is to be output, the default value of the expression.

The default error processing condition is on when the system's power is turned on.

The errors and default values are:

Error (Number)	Default Values
Underflow (1)	0
Integer precision overflow (2)	+ or − 99999
Short precision overflow (2)	+ or − 9.9999E99
Real precision overflow (2)	+ or − 9.99999999999E499
`COT` or `CSC` of n*180°; n = integer (3)	9.99999999999E499
`SEC` or `TAN` n*90°; n = odd integer (4)	9.99999999999E499
Zero ∧ negative power (5)	9.99999999999E499
Zero ∧ zero (6)	1
Uninitialized numeric variable (7)	0
Uninitialized string variable (7)	""
Division by zero (8)	+ or −9.99999999999E499

For instance, try to divide a number by zero:

```
34/0
Warning  8 : /ZERO
 9.99999999999E499
```

Beeps and displays a warning message.
Answer; default value of expression.

Since the default condition is on at power on, the system beeps and displays a warning message to alert you to the error. But the cursor moves to the next line on the display after the warning so that, essentially, the error is ignored.

The `DEFAULT OFF` statement cancels the use of default values for math errors and sets the system to normal error processing. For instance, type:

```
DEFAULT OFF
34/0
Error 8 : /ZERO
```

Sets system to normal error processing.
Try dividing by zero again.
Beeps and displays an error message.

With `DEFAULT OFF`, such an error would halt the execution of a running program.

To reset the system to default error processing, execute:

```
DEFAULT ON
```

With `DEFAULT ON`, the math errors stated above do not halt the execution of a running program.

Part II

BASIC Programming With Your HP-85

Section 5

Simple Programming

If you have read the Getting Started section of this handbook, you've already seen that by using the programming capability of your HP-85, you save hours of time in long computations.

With your HP-85 Personal Computer, Hewlett-Packard has provided you with a Standard Pac containing 15 programs already recorded on a magnetic tape cartridge. You can begin using the programming power of the HP-85 by simply using any of the programs from the Standard Pac, or from one of the other Hewlett-Packard pacs in areas like finance, statistics, mathematics, engineering, or linear programming. The growing list of application pacs is continually being updated and expanded by Hewlett-Packard to provide you with a wide variety of software support. For the advanced programmer, Hewlett-Packard will supply plug-in ROMs to give your system additional capabilities and will provide peripheral devices with the necessary interfacing.

However, we at Hewlett-Packard cannot possibly anticipate every problem for which you may want to use your HP-85. In order to get the *most* from your personal computer, you'll want to learn how to program the HP-85 with BASIC programming language to solve your every problem. This part of the *HP-85 Owner's Manual and Programming Guide* introduces the BASIC language, the editing features of the HP-85, and gives you a glimpse of just how sophisticated your programming can become with the HP-85 Personal Computer.

After most of the explanations and examples in this part, you will find problems to work using your HP-85. These problems are not essential to your basic understanding of the computer, and they can be skipped if you like. But we urge you to work them. They are rarely difficult, and they have been designed to increase your proficiency, both in the actual use of the features of your HP-85 and in creating BASIC programs to solve your *own* problems. If you have trouble with one of the problems, go back and review the explanations in the text, then tackle it again.

In programming, there is no uniquely correct program to solve a particular problem. Any solution that yields the correct output is the right one, but we have included *sample* solutions to the problems in appendix F. Thus, you'll have programs to use, modify, and enhance—even if you're a beginning programmer. In fact, when you have finished working through this part and learned all the capabilities of the HP-85, you may be able to create programs that will solve many of the problems faster, or in fewer steps, than we have shown in our illustrations.

One more thing: this handbook has been written under the assumption that most of you have had *some* programming experience. If you have never written a program before, you may wish to become more familiar with BASIC programming through the optional HP-85 BASIC Training Pac. On the other hand, many of you may be quite experienced BASIC programmers, in which case the *HP-85 Pocket Guide, HP-85 BASIC Reference Card,* and appendix D, Glossary and BASIC Syntax Guidelines, will serve you best.

Now let's start programming!

Loading a Prerecorded Program

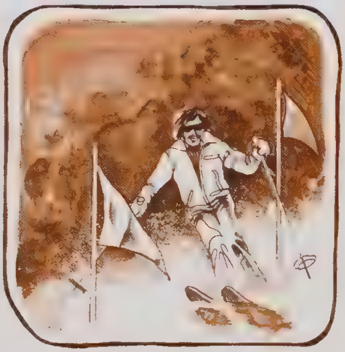
If you worked through Getting Started (pages 17 through 31), you learned how to create, enter, and record a BASIC program to compute the average of a set of numbers. Now look at a more complex program.

Insert the Standard Pac tape cartridge into the tape drive as we did earlier (page 22), printed side up, open edge toward the computer.

Next, load the Ski Game program from the Standard Pac:

1. Press the LOAD key, which displays LOAD on the CRT, or type LOAD.
2. Type the program name, enclosed within quotation marks; in this case type "SKI".
3. Press END LINE to execute the LOAD"SKI" command.

Now the system will search for the Ski Game program and load it from the tape into computer memory. The amber tape drive light will glow while the tape drive is in operation and the display will be turned off. You can easily see when a program has been loaded completely because the cursor will return to the display and the amber tape drive light will stop glowing. Once the Ski Game program has been loaded into computer memory, you can test your "skiing" skills by trying to descend a slalom ski course without missing any of the "gates."

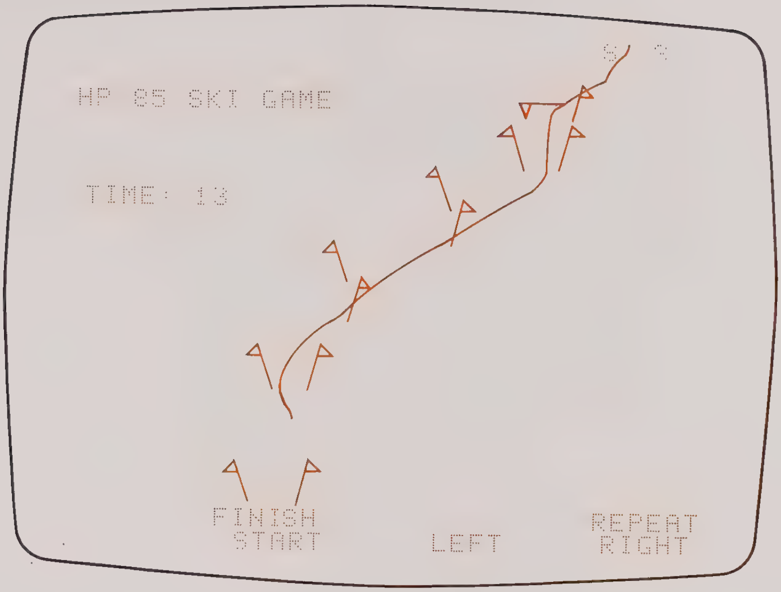


The Game. The Ski Game simulates a skier descending a slalom course, with you in control of the skis. Before you begin your descent, the program asks you whether you wish to ski on a white or a black background, asks you to enter a course code (any number) so that you can ski the same course again or try a different one by specifying a different code, and then asks for your skiing ability. As the game begins, a "skier" comes shooting down the course determined by the flags. You control the direction of the skier by tapping the special function keys k3 and k4, labelled LEFT and RIGHT.

The object of the game, of course, is to have a perfect slalom run in record time, without missing any of the gates determined by the flags. (Record time on the most difficult course is about 8.5 seconds.)

After you have loaded the program, press RUN and then press k1 (SET UP) to set up the ski course.

As soon as you press k2 (START), the game begins. You're on the slope, racing against the clock—you're in control. After each ski run, the HP-85 will display your time and missed gates. Then you can either try the same course again (by pressing the special function key corresponding to REPEAT, k8) or specify a new course (by pressing the key corresponding to START, k6).



Stopping a Running Program

Remember, *you* are in control of the HP-85. Although the Ski Game program gives you the option of stopping the game, most of the Standard Pac programs will continue to run unless you halt the program.

Stop a running program by pressing the PAUSE key or almost any other key. The program can be continued after it has been halted by the PAUSE key, by pressing CONT *(continue)*. Pressing almost any other key will halt the program *and* perform the indicated function of the key.

Listing a Program

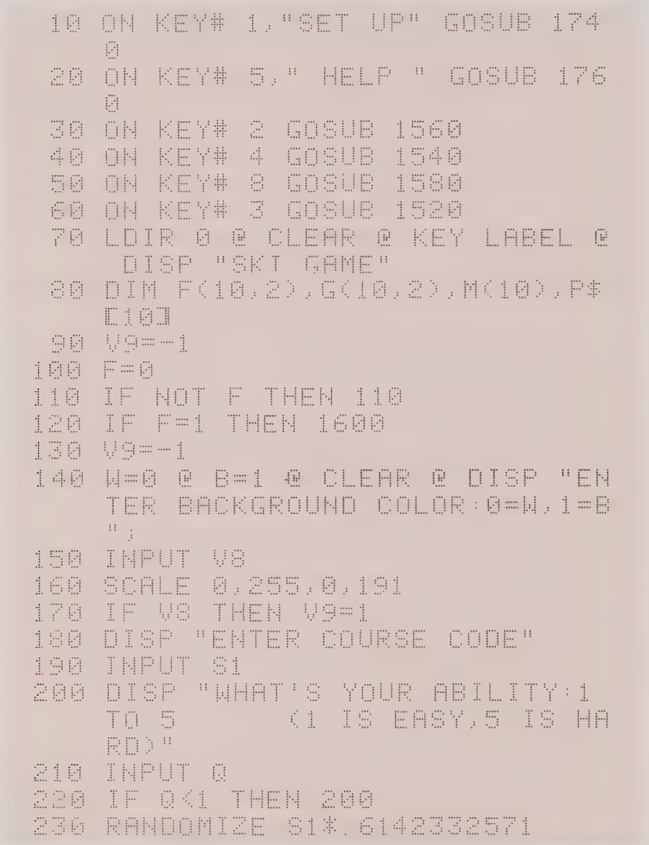
The HP-85 will give you a listing of any program contained in computer memory at any time, on the display or on printer paper. To see a listing of the Ski Game program that is now loaded in the computer memory, press the LIST key. The LIST key will stop the running program and list the first full screen of the program on the display.

Each successive time that LIST is pressed, another full screen of program lines is displayed until the end of the program is reached. Following the last line of the program, the system displays the remaining number of memory locations.

You can obtain a printed listing of the program by pressing SHIFT P LST *(printer list)*. The program will be listed in its entirety, unless you press any key to halt the listing.

Now list 20 lines or so of the Ski Game program with (P LST); your printout should look like the one shown here.

(P LST)



```
 10 ON KEY# 1,"SET UP" GOSUB 174
    0
 20 ON KEY# 5," HELP " GOSUB 176
    0
 30 ON KEY# 2 GOSUB 1560
 40 ON KEY# 4 GOSUB 1540
 50 ON KEY# 8 GOSUB 1580
 60 ON KEY# 3 GOSUB 1520
 70 LDIR 0 @ CLEAR @ KEY LABEL @
     DISP "SKI GAME"
 80 DIM F(10,2),G(10,2),M(10),P$
    [10]
 90 V9=-1
100 F=0
110 IF NOT F THEN 110
120 IF F=1 THEN 1600
130 V9=-1
140 W=0 @ B=1 @ CLEAR @ DISP "EN
    TER BACKGROUND COLOR:0=W,1=B
    ";
150 INPUT V8
160 SCALE 0,255,0,191
170 IF V8 THEN V9=1
180 DISP "ENTER COURSE CODE"
190 INPUT S1
200 DISP "WHAT'S YOUR ABILITY:1
    TO 5      (1 IS EASY,5 IS HA
    RD)"
210 INPUT Q
220 IF Q<1 THEN 200
230 RANDOMIZE S1*.614233257 1
```

(PAUSE)

The printer lists the program exactly as it appeared on the display except that the second and third lines of longer statements (like 10, 20, 80, 140, 200, etc.) appear indented under the first line of the statement, for greater readability. Also, blank lines are inserted every 60 lines (to the nearest complete line) for cutting to place lists in 11-inch notebooks.

What Is a BASIC Program?

A program is an organized set of instructions that tells the computer to accomplish certain tasks. Once a program has been written and loaded into computer memory, it can be executed as many times as you wish—usually at just the touch of the (RUN) key.

Statements

The instructions in a BASIC program are called statements. If you look at the Ski Game listing, you'll see that each statement (except assignments statements like `F=0` or `V9=-1`) contains one or more *keywords* which have a special meaning in BASIC. They identify operations to be performed *(executable statements)* or give the computer information it will need to execute other statements *(declaratory statements)*.

Here are some examples of BASIC keywords:

Executable	**Declaratory**
`PRINT`	`DIM`
`DISP`	`COM`
`IF...THEN`	`REAL`
`INPUT`	`SHORT`
`READ`	`INTEGER`
`FOR`	`REM`
`NEXT`	`DEF FN`
`GOTO`	`FN END`
`GOSUB`	`DATA`
`RETURN`	`IMAGE`
`LABEL`	`ON KEY#`
`CLEAR`	
`BEEP`	

Most executable statements can be executed from the keyboard without a statement number. Exceptions are noted.

Statement Numbers

Every statement in a program must be preceded by a unique statement number. These statement numbers can be seen on the left side of the Ski Game program listing, beginning with 10 and in increments of 10. However, statements may be numbered by any integer from 1 through 9999.

Statements are stored, by number, in ascending order. But you can type them in any order because statements are automatically sorted as they are entered.

Normal program execution proceeds from the lowest numbered statement to the highest numbered statement. The order of execution can be altered, however, as we'll see in sections 7 and 9.

The `END` statement should be the highest numbered statement in a program. It not only tells the computer *where* a program ends, but also *terminates* program execution. (You may also use the `STOP` statement; on the HP-85 both `END` and `STOP` perform exactly the same function. `END` is provided on the HP-85 for compatibility with other BASIC systems.)

Commands

A command is an instruction to the computer that is executed from the keyboard. Commands are used to manipulate programs and for utility purposes, such as listing programs and rewinding the tape. Most often, commands are not used in programs. But the HP-85 will allow you to program certain commands. (Refer to table below.)

Probably the two most important commands are `SCRATCH` and `RUN`. The `SCRATCH` command erases program memory and the `RUN` command starts executing the current program in memory. Both of these commands may be executed by pressing the key with the respective label or by typing the command name and pressing (END LINE). When you press the (RUN) key, that command is executed immediately. But when you type the command name, or press (SCRATCH) (which is a typing aid to display `SCRATCH`), it won't be executed until you press (END LINE).

Here is a table of the system commands. They are discussed in appropriate places throughout this manual.

Non-Programmable	Programmable
`AUTO`	`CAT`
`CONT`*	`COPY`*
`DELETE`	`CTAPE`
`INIT`*	`ERASETAPE`
`LOAD`	`FLIP`
`REN`	`LIST`*
`RUN`*	`PLIST`*
`SCRATCH`	`PRINT ALL`
`STORE`	`REWIND`*
`UNSECURE`	`SECURE`

Clearing Computer Memory

When you loaded the Ski Game program, the program was copied from the tape into computer memory. Before you key in a new program, you will first want to clear, or erase, the Ski Game program from the computer memory.

To clear a program from computer memory, you can either:

1. Press (SCRATCH) (END LINE) or type `SCRATCH` (END LINE). The `SCRATCH` command deletes the current program and all variables from computer memory.
2. Load another program from a magnetic tape cartridge. When you load a program into computer memory with the `LOAD` command, the system automatically clears computer memory before the new program is loaded.

Of course, whenever the system is turned off, it loses all contents of computer memory.

Now you are going to write your own program into the computer from the keyboard, so press (SCRATCH) (END LINE) to clear the HP-85 of the previous program.

Writing a Program

In Getting Started you created, entered, and ran two BASIC programs: a Pythagorean Theorem program, and an averaging program. In this section, we'll create, load, and run another program to show you how to use some of the features of the HP-85.

Before we do this, we'll define the conventions we use to describe program statements and system commands.

Conventions

`DOT MATRIX`	All items in `dot matrix` must be typed exactly as shown; but you may use either capital letters or small letters in statements or commands.
[ ]	Items within square brackets are optional unless the brackets themselves are in dot matrix.
...	Three dots indicate that the previous item can be repeated.

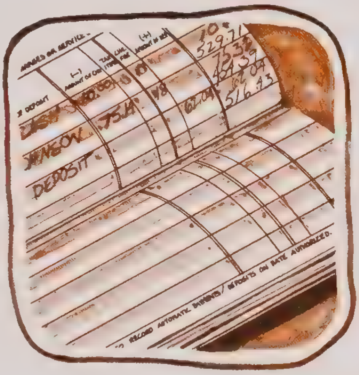
For example: `INPUT` variable name [`,` variable name ...]

---

* The keys corresponding to these commands are immediate execute keys; i.e., when you press the key, that command is executed immediately.

The statement above tells us that `INPUT` must be spelled as shown (but you can use either capital letters or small letters) and at least one variable name must be specified with the `INPUT` statement. The information within the brackets tells us that more variable names can be specified and, if they are, they must be separated by commas since the comma is in dot matrix.

Now let's write a program to keep track of a checkbook balance. Remember that in order to write a program, you must first *define the problem* thoroughly. It may help to ask yourself the following questions:



1. What answer(s) do I want?
2. What information do I know already?
3. What method will I use to find the solution from what I know?
4. How can BASIC and the HP-85 help me solve the problem?

Let's answer these questions for a simple checkbook balancing program.

1. You want to find the balance of the checking account after each check or deposit.
2. You already know the initial balance and the amount of each check or deposit.
3. You must subtract the amounts of the checks and add the amounts of the deposits to the balance.
4. Here's a sample BASIC program, as it would appear listed on printer paper.

```
 10 REM **CHECKBOOK BALANCE**
 20 REM B is the balance.
 30 REM A is the check or deposi
    t amount.
 40 DISP "Initial Balance"
 50 INPUT B
 60 DISP "Check(-) or Deposit(+)
    Amount"
 70 INPUT A
 80 LET B=B+A
 90 PRINT B ! Print new balance
100 GOTO 60
110 END
```

This is only one way to solve the problem. Can you think of other programs that would accomplish the same tasks?

Again, you can see that each statement is preceded by a number and each statement begins with a *keyword* which identifies the type of statement. For example, this program contains 11 statements: three `REM` *(remark)* statements, two `DISP` *(display)* statements, two `INPUT` statements, one `LET` *(assignment)* statement, one `PRINT` statement, one `GOTO` statement, and one `END` statement.

These keywords will be discussed individually after we execute the program.

Entering a Program

Before you enter the program, let's examine three facets of entering program statements into computer memory: automatic numbering, the spacing in program statements, and the use of the END LINE key.

Automatic Numbering

The AUTO command enables statements to be numbered automatically, as they are entered and stored, saving you the time of typing the numbers yourself.

AUTO [beginning statement number [, increment value]]

To cut your typing time further, the AUTO command is provided as a typing aid with the AUTO key; when you press AUTO, the word AUTO appears in the display. Then you can specify the beginning statement number and the increment value. If neither parameter is specified, executing AUTO causes statement numbering to begin with 10 and be incremented by 10 as statements are stored. If only the beginning statement number is specified, numbering begins at that number and is incremented by 10 as program statements are stored. Press END LINE to execute the AUTO command. For example, executing:

```
AUTO 100,5
```

Causes numbering to begin with 100 and increment by 5.

To stop the auto numbering, backspace over the unwanted numbers and type NORMAL END LINE . Auto numbering will also be halted by any executable statement or command without a number. For instance, if, after you enter the program, you wish to run it immediately, simply press the RUN key.

Spacing

In general, spacing between characters is arbitrary; the HP-85 automatically sets proper spacing into each statement of a program whenever the program is listed.

Blanks are ignored in BASIC statements except when enclosed in quotes or when contained in remarks. When the HP-85 formats a statement, blanks are inserted or deleted so that all keywords are surrounded by a blank on either side.

For example:

```
100LETA=B*C
100 LET A = B * C
1   CCLE      TA=     B*  C
```

All of the above statements are equivalent and would appear in a listing as:

```
100 LET A=B*C
```

The only place that a blank space should not be typed is immediately after the first letter of a keyword. If you attempt to enter the statement, the system will interpret the first letter as a variable name and will give you an error message. For example:

```
[illegible] L ET A=B*C
[illegible]        SYNTAX
```

Statement Length

As we mentioned earlier (page 37), program statements can be up to 95 characters long including the line number—that's three full lines on the video display minus one space to press END LINE.

But if you "pack" your statements by deleting all spaces between characters, be sure to take into account that the system will automatically insert spaces around keywords when the statement is listed or edited—the statement may be too long to be edited and reentered.

(The system will give you an error message if it does not understand.)

Entering Program Statements into Computer Memory

Program statements are entered into computer memory in the same way that any executable keyboard operation is entered, by pressing END LINE. You must press END LINE after each program statement has been typed in. Pressing END LINE also causes the statement to be checked for syntax errors before it is stored. Should an error occur on entering a statement, simply correct or retype the statement, then reenter it. Refer to section 6, Program Editing.

In a long statement that requires more than one line, do not press END LINE until the statement is completely typed in; the system display will automatically wrap around onto the next line. Press END LINE only to enter a complete program statement into computer memory.

For example:

`10 PRINT "After you type a compl`	Do not press END LINE here ...
`ete program statement, you must`	... or here ...
`enter it by pressing ENDLINE."` END LINE	... but here.
`20 END` END LINE	And here.

Entering the Program

You can enter a program into computer memory in either of two ways:

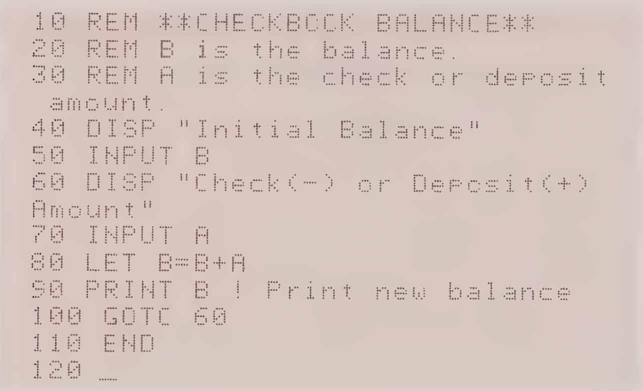
1. By retrieving a copy of a previously stored program from magnetic tape cartridge.
2. By typing the program statements, including statement numbers, one at a time from the keyboard, pressing END LINE after each statement. (Remember, you don't need to type the statement numbers if you use `AUTO` line numbering.)

Since we do not have a program on tape that keeps track of a running checkbook balance, we will use the second method to enter our program.

If you have not already done so:

1. Press SCRATCH END LINE to erase previous programs from computer memory.
2. Press CLEAR to clear the display. This is not a necessary step to writing a program, but it will increase the legibility of the display.
3. Now, press AUTO END LINE since we want to take advantage of the automatic numbering system.

Finally, enter the checkbook balancing program by typing each statement shown in the sample program, pressing (END LINE) after each statement. When you have finished entering the program, the display will look like this:



```
10 REM **CHECKBOOK BALANCE**
20 REM B is the balance.
30 REM A is the check or deposit
 amount.
40 DISP "Initial Balance"
50 INPUT B
60 DISP "Check(-) or Deposit(+)
Amount"
70 INPUT A
80 LET B=B+A
90 PRINT B ! Print new balance
100 GOTO 60
110 END
120 _
```

The program for keeping track of a checkbook balance is now loaded into computer memory. Notice that AUTO statement numbering caused the number 120 to appear below the END statement of the program. Simply backspace over 120 to erase the number, and type NORMAL (END LINE) to stop AUTO statement numbering.

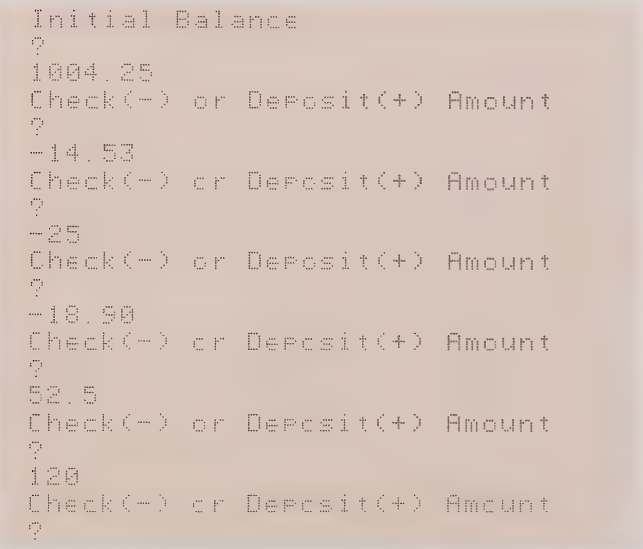
Running a Program

To run a program, you have only to press the (RUN) key.

For example, use the program now in computer memory to balance a checkbook with an initial balance of $1,004.25; checks written for the amounts of $14.53, $25.00, and $18.90; and deposits in the amounts of $52.50 and $120.00

(RUN)

Press (RUN) to start program execution.



```
Initial Balance
?
1004.25
Check(-) or Deposit(+) Amount
?
-14.53
Check(-) or Deposit(+) Amount
?
-25
Check(-) or Deposit(+) Amount
?
-18.90
Check(-) or Deposit(+) Amount
?
52.5
Check(-) or Deposit(+) Amount
?
120
Check(-) or Deposit(+) Amount
?
```

When a question mark appears, key in the balance and press (END LINE). Then enter the checks as negative numbers and the deposits as positive numbers. Press (END LINE) after the amount to enter the data to the program.

(PAUSE)

Press (PAUSE) to halt program execution.

Where is the record of the checkbook balance? You'll find that the printer has recorded the following on paper. (Press the PAPER ADV key to advance the paper if necessary.)

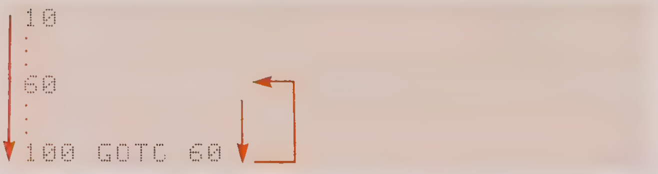
```
589.72
564.72
945.82
998.32
1118.32
```

Initial balance − 14.53.
New balance − 25.00.
New balance − 18.90.
New balance + 52.50.
New balance + 120.00.

Now let's see how the HP-85 executed this program.

Order of Program Execution

Statements are executed in order of ascending statement numbers.



```
10
:
60
:
100 GOTO 60
```

When you pressed RUN, the HP-85 began executing instructions sequentially by statement number starting with statement `10`. When it reached statement `100 GOTO 60`, the system returned to statement `60` and executed successively higher-numbered statements from there. The program continued to run until you pressed PAUSE to halt the program.

Fundamental BASIC Statements

Now let's examine the statements that composed our checkbook balancing program.

REMarks

Many times you may want to insert comments in order to make your program logic easier to follow. This can be done by using the `REM` *(remark)* statement or `!`, the comment delimiter.

`REM` [any combination of characters]

In our sample program, remarks are used to remind us that the variables A and B stand for amount of a check or a deposit and the checkbook balance, respectively:

```
10 REM **CHECKBOOK BALANCE***
20 REM B is the balance.
30 REM A is the check or deposit
   amount.
```

The comment delimiter, `!`, can be anywhere in a program statement after the statement number. All characters following a `!` are considered part of a comment unless the comment delimiter, `!`, is within quotes.

In this way, program statements can contain comments. For instance, statement `90` in our sample program:

```
90 PRINT B ! Print new balance
```

Comments, as you have seen, are useful only in a program listing. They do not affect program execution.

DISPlay

The `DISP` *(display)* statement allows text and variables to be output on the display.

`DISP` [display list]

The display list can contain variable names, numeric expressions, quoted text or messages, and the `TAB` function (covered later). These items must be separated by commas or semicolons.

In the checkbook balancing program, the following `DISP` statements appear:

```
40 DISP "Initial Balance"
:
:
60 DISP "Check(-) or Deposit(+)
   Amount"
```

As you have seen, when these statements are executed in a running program, they display, respectively:

```
Initial Balance
:
:
Check(-) or Deposit(+) Amount
```

You can combine quoted messages with variable names, but they must be separated from each other with commas or semicolons. For example:

```
n=5
S=175.60
DISP "THE AVERAGE OF THE";N;"NUM
BERS IS";S/N
THE AVERAGE OF THE 5 NUMBERS IS
 35.12
```

When these statements are executed ...

... this message is displayed.

What is the difference between using commas or using semicolons to separate items in a display list? Look at the following examples:

```
DISP 111,222,333,444,555,666
 111                 222
 333                 444
 555                 666
```

Commas cause wide spacing between display list items.

```
DISP 111;222;333;444;555;666
 111  222  333  444  555  666
```

Semicolons space items close together.

Notice the difference in spacing between the items. When an item is followed by a comma, the next item will be left-justified at either column 1 or column 22 on the display. Remember, every number has a leading blank or a minus sign and a trailing blank for spacing. If a number contains over nine digits and would start in column 22, it will be displayed in the first column of the next line.

When an item is followed by a semicolon, no additional blanks are inserted. For example:

```
DISP 100; -20; 77.3
 100 -20  77.3
```

All numbers are displayed with a leading blank or minus sign and a trailing blank for spacing.

Two or more commas after an item cause one or more character fields to be skipped.

For example:

```
DISP ,100,,200
                          100
                          200
DISP 100,,200
 100
 200
```

When a `DISP` statement appears without a display list, a blank line is displayed. For example:

```
 10 DISP
 20 DISP
 30 DISP "*********************"
 40 DISP "* SQUARINUT SOUBISE *"
 50 DISP "*********************"
 60 DISP
 70 DISP
 80 DISP "Peel and mince 3 large
     onions and set them aside."
 90 DISP "Slowly add 4 Tbsp. flo
    ur to 1/4 cup squarinut oil.
    "
100 END
```

When this program is executed, the following would be displayed.

```
*********************
* SQUARINUT SOUBISE *
*********************


Peel and mince 3 large onions an
d set them aside.
Slowly add 4 Tbsp. flour to 1/4
cup squarinut oil.
```

When the display list ends with a comma or semicolon, any future `DISP` statement output is appended to the current display line. For example:

```
10  DISP "ENTER DATE"
20  INPUT D1$
30  DISP "TODAY IS ";
40  DISP D1$
50  END
```

When these statements are executed in succession in a program …

```
ENTER DATE
?
JUNE 1
TODAY IS JUNE 1
```

… and you enter JUNE 1 for the date …
… this message is displayed.

The semicolon at the end of the statement 30 causes the message "TODAY IS" to be held in a special disp/print buffer. The buffer does not display (or with `PRINT`, print) its contents unless:

- Another `DISP` statement without a semicolon at the end of the message causes it to be output.

- An `INPUT` statement causes the buffer contents to be displayed or printed (as we'll see later).
- The buffer is filled with 32 characters, in which case it is automatically output.

For instance, if statement 40 also ends with a semicolon (in the example above), an extra `DISP` statement is required to output the message:

```
10  DISP "ENTER DATE"
20  INPUT D1$
30  DISP "TODAY IS " ;
40  DISP D1$;
50  END
```

If you run this program, the input prompt will be displayed and nothing else *appears* to happen.

```
ENTER DATE
?
JUNE 1
DISP
TODAY IS JUNE 1
```

Enter the date.
If you now execute `DISP`, the message will be displayed.

The extra `DISP` statement could also have been part of the program between statements 40 and 50.

PRINT

The `PRINT` statement allows text and variables to be printed by the HP-85's internal printer.

`PRINT` [print list]

Like the display list, the print list can contain variable names, numeric expressions, string expressions and quoted text, and the TAB function. All items must be separated by commas or semicolons.

Here are some examples:

```
PRINT 20;81.1569;32.9
PRINT "HYPOTENUSE=";5
PRINT "***!!!";"///^^^";"@@@***"
PRINT
PRINT
PRINT "***!!!","///^^^","@@@###"
PRINT
I,J,K,L,M=5
PRINT I,J,K,L,M
```

```
 20  81.1569  32.9
HYPOTENUSE= 5
***!!!///^^^@@@***


***!!!                  ///^^^
@@@###

 5                       5
 5                       5
 5
```

Notice that commas and semicolons perform in the `PRINT` statement just like in the `DISP` statement. A comma after an item causes the next item to be left-justified in either column 1 or column 22. A semicolon after an item suppresses additional blanks. Also note that when nothing follows the word `PRINT` in a statement, the paper advances one line. For more information about displaying and printed output, refer to section 10, Printer and Display Formatting.

INPUT: Assigning Values From the Keyboard

The `INPUT` statement allows values in the form of expressions to be assigned to variables from the keyboard at the request of a program. The `INPUT` statement is programmable only; it can't be executed from the keyboard.

`INPUT` variable name$_1$ [, variable name$_2$ ...]

As we have seen, when the `INPUT` statement is executed, a question mark (`?`) appears on the display. A value can then be input for each variable designated in the `INPUT` statement.

Remember our first example in section 1 (page 27):

```
60 INPUT L,W
```

The program called for the lengths of the sides of a right triangle.

When the program was executed, and the question mark appeared on the display, we input both values, separated by a comma, in one line like this:

```
?
7.5, 10 END LINE
```

Separate `INPUT` values with commas.

If we had tried to enter the values for the variables L and W one at a time, we would have received an error message, followed by another question mark and we'd have another chance to enter *both* values. An `INPUT` statement requests all values for the variables specified to be entered at the same time.

Values for string variables can be quoted or unquoted, but an unquoted string cannot contain a comma (since commas separate input items).

Let's look at some examples of entering strings:

```
60 DISP "YOUR NAME";
70 INPUT N$
80 DISP "MY NAME IS"; N$
```

When these lines are executed, the display shows:

```
YOUR NAME?
```

Now you can input up to 18 characters of your name in either of two ways;

Without quotation marks:

```
YOUR NAME?
HP-85
MY NAME ISHP-85
```

Since you did not leave any trailing blanks in `DISP` statement 60, then `DISP` statement 80 packs the characters together.

With quotation marks:

```
YOUR NAME?
"  HP-85!"
MY NAME IS HP-85!
```

Use quotation marks if you wish to preserve leading or trailing blanks or use commas in your expression.

Whenever you assign character string expressions to string variables from the keyboard, you can use quotation marks at your option. Just remember that strings do not contain leading or trailing blanks unless you specify them explicitly with quotation marks.

Also notice where the question mark appeared in the examples above. If you place a semicolon after a message in a `DISP` or `PRINT` statement before an `INPUT` statement, the semicolon suppresses the carriage return so that the question mark appears on the same line as the message.

Thus, we could have written our checkbook balancing program like this:

```
40 DISP "Initial Balance";
50 INPUT B
60 DISP "Check or Deposit Amount";
```

Before an `INPUT` statement, a semicolon at the end of a `DISP` statement (or `PRINT` statement) suppresses the carriage return.

If this section of the program were executed, it would display:

```
Initial Balance?
.
.
Check or Deposit Amount?
```

Question marks are on the same line instead of beneath the displayed line.

Pressing (END LINE) without entering values when numeric input is requested causes an error. You may input the null string ("") as response to a string input request.

BEEP

The BEEP statement can be considered a comment or an output statement. The BEEP statement is used to produce an audible tone of variable frequency and duration that can be used in a number of ways.

BEEP [tone , duration]

BEEP can signal that a particular computation or program segment is complete. It can be used to indicate audibly that the computer is ready for input, so that the operator does not have to remain at the keyboard. And, of course, it can be used for the sound itself; load the COMPZR program from the Standard Pac—you can actually compose "music" with the BEEP statement.

If no parameters are specified, the frequency is approximately 2000 hertz, and duration is 100 milliseconds. By specifying parameters, you can change the tone and the duration.

For example, we used BEEP in the hypotenuse program as an audible input prompt:

```
10 DISP "ENTER SIDE LENGTHS"
20 DISP "OF A RIGHT TRIANGLE,"
30 DISP "SEPARATED BY A COMMA."
40 DISP "THEN PRESS END LINE."
50 BEEP
60 INPUT L, W
70 D= SQR(L^2+W^2)
80 PRINT "HYPOTENUSE =";D
90 END
```

BEEP signals the operator for input.

The BEEP statement can be executed from the keyboard. For example, try several different values for tone and duration by executing the following statements. You can stop the sound at any time by pressing (STEP PAUSE).

```
BEEP
BEEP 10, 50
BEEP 200, 30
BEEP 50, 100
```

The value for the tone and duration of BEEP can be a numeric expression. *Both parameters are rounded to integer values with the* BEEP *statement.* For example, run the following program to generate "random" music. (We discuss the FOR and NEXT statements in section 7). You can stop the program at any time by pressing (PAUSE).

```
10 FOR I=1 TO 250
20 BEEP I*RND+1, 50
30 NEXT I
40 END
```

This program generates a "random" sequence of 250 audible tones.

Use the following formulas to compute `BEEP` parameters that produce a particular frequency and duration:

Tone (first parameter):

`P1 = 613062.5/(11*F)-134/11` where `F` is desired frequency in hertz.

Duration (second parameter);

`P2 = T*613062.5/(11*P1+134)` where `T` is desired duration in seconds. (Or, simply, `P2 = T*F` when `F` is known.)

For example, to `BEEP` for approximately one-half second at a frequency of about 440 hertz, compute `P1` and `P2` as follows:

```
P1 = 613062.5/(11 * 440)-134/11
P1
 114.483987603
P2 = .5 * 440
P2
 220
```

So, the `BEEP` statement would be:

```
BEEP 114,220
```

Beeps at approximately 440 hertz for approximately 0.5 second.

LET: Assignments

Any numeric variable can be assigned a value using an assignment statement as we have seen in section 3. String variables can also be assigned string expressions using the assignment statement if the expression produces a string shorter than or equal in length to the size of the string variable. The keyword in an assignment statement is `LET`, but its use is optional.

[`LET`] simple variable$_1$ [, simple variable$_2$...] = numeric expression

[`LET`] string variable$_1$ [, string variable$_2$...] = string expression

The keyword `LET` is a reminder that the variable name is always to the left of the equals sign and the expression assigned to that variable is always to the right of the equals sign; you are ``letting`` the variable be changed to equal the value of the expression.

For example, the following statements are equivalent:

```
X=12
LET X=12
X = 3*4
```

The following statements using string variables are equivalent:

```
A$, B$="BUTTERFLY"
LET A$,B$= "BUTTER" & "FLY"
```

Remember that the string expression must be enclosed within quotes in a string variable assignment statement.

To check the current value of a variable, type in its name, then press END LINE. For instance, using the above values for the variables:

```
X
 12
A$
BUTTERFLY
B$
BUTTERFLY
```

Pressing END LINE after the variable name yields its current value.

If a numeric variable is used in a computation and hasn't been assigned a value, a warning message is displayed and 0 is used as its value. Likewise, if a string variable is used before being assigned a value, a warning message is displayed and the null string is used as its value. In general, it is good programming practice to initialize variables (by initialize, we mean assign them their initial values) at the beginning of the program, as we did in the averaging program (page 29).

GOTO: Unconditional Branching

In our checkbook balancing program, you saw that the `GOTO` statement transferred program control back to the specified statement. This is known as an unconditional branch. `GOTO` statements are programmable only; they can't be executed from the keyboard.

`GOTO` statement number

If the specified statement is not an executable statement (e.g., a `REM` statement), control is transferred to the first executable statement following that statement.

As you may remember from the checkbook balancing program, the use of the `GOTO` statement caused the program to "loop" endlessly from statements 60 through 100:

```
60 DISP "Check(-) or Deposit(+)
Amount"
70 INPUT A
80 LET B=B+A
90 PRINT B
100 GOTO 60
```

Branches to statement 60.

But we also saw that it is easy to stop the program—by pressing PAUSE.

`GOTO` statements may branch to both higher numbered and lower numbered statements; for example:

```
10 X=5
20 GOTO 50
30 DISP "NEW X-VALUE";
40 INPUT X
50 PRINT "X EQUALS";X
60 GOTO 30
```

Branches to statement 50.

Branches back to statement 30.

The `GOTO` statement is the most simple form of branching.

Multistatement Lines

A symbol that you may have seen in the Ski Game program listing is the "at" symbol (`@`). The `@` symbol enables you to type more than one statement on the same program line, thus shortening program listings and conserving memory. Remember you still cannot enter more than 95 characters (including the statement number) at a time.

Examine line 70 in the Ski Game program listing:

```
70 LDIR 0 @ CLEAR @ KEY LABEL @
    DISP "SKI GAME"
```

In the program line above, four statements have been joined together on the same program line, using the same statement number. The program could have been written like this:

`70 LDIR 0`	Sets label direction.
`73 CLEAR`	Clears display.
`75 KEY LABEL`	Recalls key labels.
`77 DISP "SKI GAME"`	Displays message.

But, by using the `@` symbol, the program was shortened by three program lines (nine bytes).

There are several things you must be careful about when you type multiple statements using the same statement number.

- If there is a `GOTO` statement in a multistatement line, it should be the last statement. For instance:

  ```
  20 GOTO 50
  .
  .
  50 GOTO 60 @ PRINT "MISSED"
  60 END
  ```

  In order to reference the print statement in line 50, the statements need to be reversed; otherwise, the message will not be printed.

- If you join statements that involve relational tests or "decision-making" operations (like `IF ... THEN`) be sure that you are aware of what happens when the test comes up "true" or "false." For instance if you join a statement at the end of an `IF ... THEN` statement, that statement will not be executed if the relational test is false. This can be a definite programming advantage, but there may be times when you forget this fact and wonder why your program doesn't work the way you think it should.

- Declarations (such as `DIM`, `COM`, `REAL`, `SHORT` and `INTEGER`) can be made in multistatement lines but they must be the last statement in the line.

- Anything that follows `REM` or `!` is a remark. The following multistatement line may look good, but `D` will never be computed!

  ```
  50 R = 4.5 ! RADIUS @ D = 2*R
  ```

- Programs that have multistatement lines using the `@` symbol may not be transportable to other computers using BASIC.

- Care should be taken to preserve readability with multistatement lines. For instance, `CLEAR @ KEYLABEL` is easily read and understood on one line. But it is possible to destroy readability by packing too much into a line. Readability is important, particularly with debugging procedures and documentation of your program.

Problems

5.1a. Write a program to convert a temperature in Celsius degrees to Fahrenheit according to the formula $F=1.8*C+32$. Use a LET assignment statement for the conversion calculation. The program should ask for the original Celsius temperature and label the corresponding Fahrenheit temperature.

b. Write a second program to convert a temperature in Fahrenheit degrees to Celsius. The equation is $C=5/9*(F-32)$. Do not use the optional keyword LET in this program. Be sure to include an input prompt and an output label.

5.2 Janey Dair enjoys dropping her new Rebounder ball from the window of her room, delighting her friends who watch it bounce on the pavement below. Each rebound reaches a height equal to 65% of its previous height. Write a program that requests the height from which she drops her Rebounder, and displays the cumulative distance it has traveled each time it touches the pavement. Use a BEEP to represent each bounce prior to displaying the distance. The program should continue calculating the distances until it is manually interrupted with the PAUSE key. Observe the output to be sure that the total distance traveled approaches a limit, rather than increasing indefinitely.

5.3 In preparation for writing your first novel, you want to use the HP-85 to help you choose an interesting title. You decide to write a program that takes a noun and a proper noun as input, and prints two titles using the following forms:

THE (noun) OF (proper noun)
TO (proper noun) WITH THE (noun)

You may not win any literary awards, but you'll get some interesting titles.

5.4 World-famous jazz artist Bertha Blues wants to program the HP-85 to play a particular bass rhythm as an accompaniment for a work session. The rhythm consists of the repeated sequence of notes C130.81,G196, G98,G196 at 120 beats per minute (0.5 seconds per note). (The numbers following the notes specify the frequency in hertz.) Write a program that computes and prints the tone and duration parameters for the three appropriate BEEP statements and then breaks into its rhythmical rendition.

5.5 The factorial function ($x!$) is defined for positive integer values of x as

$$x! = x(x-1)(x-2)\ldots1.$$

An algebraic approximation is given by the equation

$$x!=e^{-x}x^{x}\sqrt{2\pi x}.$$

Write a program that, for any positive integer value of x, calculates and prints the $x!$ approximation using this method. (In section 7 you will see an easy method to compute the *exact* factorial function.)

5.6 During his spare time, Artemas Horologos repairs watches in his home workshop. He has decided that a program that calculates his bill would be very helpful. Write a program that requests the customer's last name, the number of hours Artemas has worked on a watch, and his cost for replacement parts. It should then print an individualized repair bill itemizing the charges for parts, for labor, and the total amount due. Artemas charges $8.50 per hour for his labor, and charges 10% more than his cost for parts.

Section 6

Program Editing

Often you may want to alter or add to a program that is already loaded into computer memory. The HP-85 has been designed to make program editing as fast and easy as possible.

In this section, we will discuss program modification by adding, deleting, and editing program statements. And we'll introduce specific program editing commands to delete blocks of program statements, list specific parts of a program, and automatically renumber a complete program. Finally, we'll show how to interrupt the execution of a running program and how to continue execution at a specified statement.

Editing Program Statements

Edit program statements in the same way that you edit anything that appears on the display—with the display editing keys.

There are two ways to edit and change a statement that is already in computer memory.

1. Recall the program into the display by using the ROLL key or by listing the program on the display. Then using the display editing keys and cursor control keys, move the cursor to the desired statement, make the necessary changes in the program statement, and press END LINE to enter the changed statement into memory.
2. Retype the statement, including statement number, incorporating all the changes you wish to make. Then press END LINE to enter the statement into memory.

Remember, you can enter program statements in any order—the computer automatically sorts them by statement number as they are entered. The last statement entered with a given statement number is the one that is used in the program. When you edit a line or statement on the display, always check to see that there are no unwanted characters beyond the last character in the statement. If there are, move the cursor to the end of the good line and press -LINE to delete the unwanted characters before you press END LINE to enter the program statement.

Deleting Statements

You can delete program statements in either of two ways:

1. To delete an individual statement of a program, type the statement number and then press END LINE.
2. To delete a section of a program, it is quicker to use the DELETE command.

The DELETE command is used to delete a statement or a block of statements from a program.

DELETE first statement number [, last statement number]

The DELETE command is provided on a key as a typing aid. When you press the DEL key, DELETE is displayed.

If only one statement number is specified with the DELETE command, then only that program statement will be deleted from program memory. If you specify both statement numbers, then that section of a program will be deleted.

Examples:

`30` END LINE	Deletes statement 30.
`DELETE 40` END LINE	Deletes statement 40.
`DELETE 60,90` END LINE	Deletes statements 60 through 90, inclusive.

Adding Statements

Add new statements to a program merely by typing and entering them into computer memory. Be sure that the statement number of a new statement positions it correctly in the program.

Often, it saves a good deal of typing by merely editing a similar statement of your program, changing the statement number, and then entering the new statement into program memory by pressing END LINE. But don't forget to change the statement number!

Renumbering a Program

The `REN` *(renumber)* command is used to renumber a program that has already been entered into computer memory.

`REN` [beginning statement number [, increment value]]

Just as with the `AUTO` command, you can optionally specify the new starting statement number and the increment between statement numbers. If no parameters are specified, the program is renumbered so that statement numbering begins with 10 and is incremented by 10. If no increment value is given, the statement numbers will be incremented by 10.

Examples:

`REN`	Renumbers a program so that the first statement is numbered 10, and the statements that follow are numbered in increments of 10.
`REN 100`	Renumbers a program, beginning with 100 and incrementing by 10.
`REN 200,5`	Renumbers a program, beginning with 200 and incrementing by 5.

The `REN` command automatically renumbers an entire program, including any branches within a program. But the `REN` command will not change the parameters of the `PLIST` or `LIST` commands when they are included as program statements.

If you have a very large program or you use `REN` in such a way that the computer reaches line 9999 before it renumbers the whole program, then the computer will automatically start at the beginning of the program and renumber by 1, i.e., beginning with statement 1 and renumbering in increments of 1.

Listing a Modified Program

Up to this point, we have discussed two ways to list a program: by using the LIST key to list the program on the display, or by using the P LST key to list the program on the printer.

But you can also type these commands from the keyboard and then specify the section of a program you wish to have listed.

```
LIST [beginning statement number [ , ending statement number]]
PLIST [beginning statement number [ , ending statement number]]
```

If you type `LIST`, and specify one statement number before pressing END LINE, listing begins with that statement and continues for one screen. If two statement numbers are specified, that section of statements between and including the two numbers is listed.

If you type `PLIST` and specify one statement number before pressing END LINE, the program will be listed on the printer from that statement number to the end of the program (or until you press a key). If two statement numbers are specified, that section of the program is listed on the printer.

If you type either command and specify no statement numbers, and then press END LINE, the command will be executed as if you had pressed either the LIST or the P LST key.

Examples:

`LIST 40,90`	Lists statements 40 through 90 on the display.
`LIST 90`	Lists statements beginning with 90 and continuing for one screen's worth of statements.

If the system cannot find the statement number, it will list the next higher statement up to the last statement number you specify. For instance, if your program is numbered from 10 to 150 in increments of 10:

`PLIST 5,45`	Lists statements 10 through 40 on the printer.

You can list one statement by specifying the same statement number for both parameters. For example:

`PLIST 90,90`	Lists statement 90 on the printer.

Both the `LIST` and `PLIST` commands are programmable. However, `REN` will not renumber programmed `LIST` or `PLIST` parameters.

One more function is associated with the `LIST` and `PLIST` commands: *following the list of the last program statement, the remaining number of memory locations (bytes) is output.* We'll discuss the system memory in section 8. For now, simply note that the number of the end of an entire program listing gives the available memory.

Interrupting Program Execution

Pausing

We have already seen that pressing PAUSE halts the execution of a running program. But actually it just suspends the execution of a running program. When you press PAUSE the current statement is completed and the program is paused at the next statement to be executed.

As we shall see, a pause can also be programmed using the `PAUSE` statement.

Although the specific function of PAUSE is to suspend the execution of a running program, pressing any key (except those noted below) will also halt the execution of a running program and perform the indicated function of the key.

For instance, if you press a typewriter key, such as C, the system finishes executing the current statement, then halts and displays "C."

But if you happen to press RUN during the execution of a running program, the current statement is completed, the program is halted, and then the system displays RUN_. If you really want to rerun the program, execute RUN_ by pressing END LINE. If you do not want to rerun the program press CONT to continue (see below).

Note: Whenever a running program is interrupted from the keyboard, the system beeps.

The following keys will perform the indicated functions *without halting the execution of a running program* or otherwise interrupting or disturbing the program:

COPY	Copies the current display to the printer.
PAPER ADV	Advances the paper in the printer.
KEY LABEL	Recalls special function key labels (if any).
GRAPH	Sets display to graphics mode.
CLEAR	Clears alphanumeric display.
▲ ROLL ▼	Rolls display contents up or down.

Continuing

If a program has been halted with the STEP PAUSE key or a PAUSE statement, it can be resumed from where it was halted by pressing the CONT *(continue)* key or by executing the CONT command. You can press CONT or execute the CONT command after almost any other program halt—as long as you have not deallocated the program. (A program would be deallocated if, for instance, you edited the program. You would then need to initialize the program, as we will see on the next page, before continuing.)

CONT [statement number]

The CONT key is an immediate execute key. Thus execution of a halted program is immediately resumed when you press the key.

You can continue program execution at a specific statement by typing CONT followed by the statement number and then pressing END LINE. For example:

`CONT 90`	Continues program execution at statement 90.

Execution of a paused program can also be restarted at the beginning with RUN (or RUN), by executing CONT 0, or by INIT CONT.

Whenever program execution has been paused, you can perform any normal keyboard activities. For instance, you can list the program in memory or perform some arithmetic calculations. And when you press CONT, program execution resumes from where it paused (unless, of course, you have cleared the program from memory by executing SCRATCH or LOAD).

Initializing a Program

The (RUN) key (or RUN command) automatically initializes a program before running it. By "initialize" we mean that the system allocates memory to all program variables, sets (initializes) variables to undefined values, and sets the program pointer to the first statement of the program.

RUN [statement number]

As with the CONT command, you can optionally specify the starting statement by typing RUN followed by the statement number and pressing (END LINE). For example:

```
RUN 100
```

Initializes and then runs a program beginning with statement 100.

If a program has been halted with a PAUSE command, computer memory remains allocated and the program pointer is set to the statement after the one it has just executed. Pressing (CONT) (or executing CONT) does not allocate or initialize program variables again. Execution merely resumes from where it left off.

If, for instance, you *edit* a program statement after you PAUSE the program, program variables are no longer allocated and the program cannot be continued with (CONT) or CONT. You must initialize the program and reallocate memory for variables by pressing the (INIT) key (or by executing the INIT command) before you press (CONT). Execution resumes from the beginning of the program—not from where PAUSE halted it.

The INIT command allocates memory to all program variables, initializes variables to undefined values, and resets the program to begin executing from the lowest numbered statement. Using (INIT) and (CONT) together performs the same function as (RUN).

Using PAUSE in a Program

The PAUSE statement can also be used in a program, as we mentioned earlier. Program execution is halted whenever the PAUSE statement is encountered in a program. The PAUSE statement does not cause the system to beep when it is halted.

Pausing is useful to control program execution. Continue a program halted by PAUSE with (CONT) (or CONT).

For example, enter the following program:

```
10 REM [illegible] VALUE [illegible]
20 N=1
[illegible]
```

Set N (number of years) to 1.
Ask for input.
Input present value.
Print value.
Ask for input.
Input interest rate.
Print value.

Calculate future value truncated to hundredths.
Print year and amount.

Pause.
Go back to 100.

Now run the program with a present value of $1000 and interest of 6% (.06).

```
RUN
Present Value?
1000                 Present Value = 1000
Interest rate?
.06                  Interest Rate = .06
                     Future Value After
                     Year 1 is 1060
CONT                 Year 2 is 1123.6
CONT                 Year 3 is 1191.01
CONT                 Year 4 is 1262.47
```

Press CONT to continue after the PAUSE in statement 130.

Here, PAUSE and CONT enable you to print one line at a time.

Delaying Program Execution

The WAIT statement is used to program a delay between the execution of two program statements.

WAIT number of milliseconds

The WAIT parameter can be any number within the range of the HP-85 but the minimum wait is 0 and the maximum wait is about 27 minutes (1,666,650 milliseconds). A negative number specifies a zero wait. The WAIT statement can be interrupted by PAUSE or almost any other key.

For instance, if you changed statement 130 in the Future Value program to:

```
130 WAIT 3000
```

The program would wait 3 seconds (3000 milliseconds) before it printed each future amount.

Error Messages

There are three types of errors that can occur during the development and execution of your program: ''syntax'' errors, ''semantic'' errors, and ''run-time'' errors.

Syntax errors may include such errors as a missing operator, a misspelled keyword, or an illegal constant or variable name. When you press END LINE after typing in a statement, it is immediately checked for syntax errors. If the statement contains no syntax errors, it is accepted and loaded into computer memory. If the statement contains a syntax error, an error message is displayed and the cursor is positioned below the first character at which the system detected an error.

You can correct the statement by inserting, deleting, or replacing characters as shown in section 2. If the error is not corrected, the statement will not be stored as part of the program, but no other harm is done. Move the cursor down the display to enter another statement or clear the line in which the error occurred.

The second type of error, a "semantic" error, occurs when you have finished loading the program into computer memory and you try to run it. Before the HP-85 attempts to run your program, it checks to verify that your program "makes sense." Semantic errors include errors such as a missing statement, duplicate user-defined functions, illegal array dimensions, etc. You are informed of all such errors before the program can be run. These errors can usually be corrected by adding, deleting, or correcting statements; they are not difficult to find because the system alerts you to them as soon as you try to run the program.

The third type of error occurs when the program is running. All "run-time" errors interrupt a running program and cause it to halt unless `DEFAULT ON` is in effect or you override the errors with an `ON ERROR` statement. With `DEFAULT ON`, the first eight errors listed in appendix E cause a *warning* message to be output, but program execution will not be halted; all other run-time errors cause the program to halt and an error message to be displayed. With `DEFAULT OFF`, all run-time errors halt program execution and display an error message.

Run-time errors can include referencing a nonexistent array element, attempting to use uninitialized data, `READ-DATA` variable mismatch, trying to write to a nonexistent file, etc.

Refer to section 13, Debugging and Error Recovery, for information on recovering from run-time errors.

Refer to appendix E for a complete list of error numbers and messages.

Problems

6.1 For problem 5.2 in the previous section, you wrote a program that computed the distance traveled by Janey Dair's Rebounder ball. If each rebound reaches a height of 65% of its previous height, the time interval to the next bounce is $\sqrt{0.65} = 0.806 = 80.6\%$ of the previous time interval. Enter your original Rebounder program, and then modify it to incorporate a `WAIT` statement that causes a delay between bounces according to the ratio given above. Let the interval between the first two bounces be 3 seconds(3000 milliseconds).

6.2 To illustrate the effect of an unbalanced force pulling sidewards on a moving object, physics teacher Millie Graham has devised a simple experiment. She fastens a string to the side of a 350-gram miniature rocket sled and secures the other end to a fixed pivot. When ignited, the rocket sled accelerates at the rate of 30 centimeters per second per second. As the sled accelerates, the string continually pulls it sidewards, and this imbalance causes it to move in a circle of radius r (centimeters), equal to the length of the string. Ms.Graham knows that the magnitude of this force f, exerted inwards on the sled by the string, is given by

$$f = \frac{350(30 \cdot t)^2}{r}$$

where t is the time (in seconds) from when the rocket is ignited. The force, expressed in dynes, can be converted to pounds by multiplying by 2.25×10^{-6}. Write a program for Ms.Graham that requests the length of the string r and then prints the force exerted by the string (in dynes and pounds) at intervals of 1 second. Have the program halt execution after each second's output so that she can determine if the string's breaking strength (2.22×10^{6} dynes, 5.0 pounds) would be exceeded. Run the program for various string lengths, using trial and error to find the shortest length (approximately) for which the string lasts at least 10 seconds.

Section 7

Branches and Loops

Normal program execution is in sequential order from the lowest numbered statement to the highest numbered statement. As we have seen with the `GOTO` statement, branching alters this process by transferring control to a statement that is not in the sequential flow.

Branches, loops, and subroutines are three methods of altering the normal flow of program execution. This section covers unconditional branching with the `ON ... GOTO` statement, conditional branching with `IF ... THEN ... ELSE`, and a method of forming efficient loops with the `FOR` and `NEXT` statements. In section 9 we will continue our discussion of branching with subroutines, the special function keys, and user defined functions.

Most of the programs we have discussed to this point have contained *unconditional branches* using the `GOTO` statement. The `GOTO` statement is simple and direct; it transfers program control to the statement number that you specify. A `GOTO` statement used in this way is known as an *unconditional branch* because it *always* branches execution from the `GOTO` statement to the specified statement number. Now you will see how to use `IF ... THEN` statements—branching that depends on the outcome of the test.

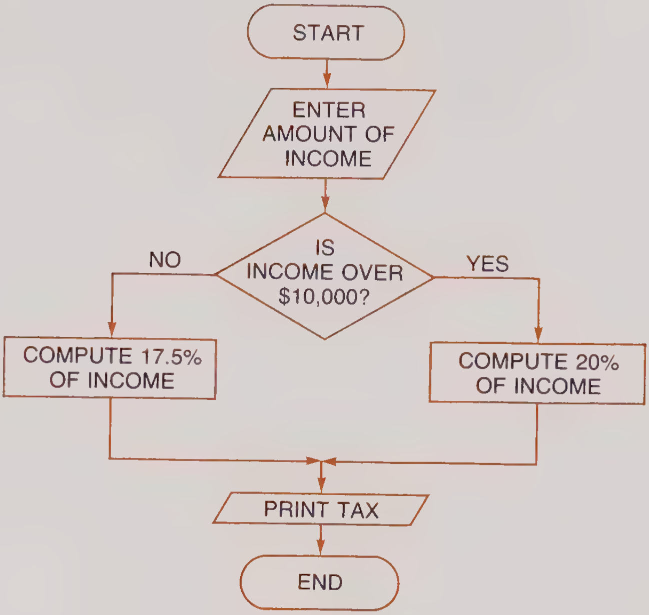
Conditional Branching

Often there are times when you want a program to make a decision. In the averaging program in section 1, we wanted the program to decide whether to branch to the end of the program to display the result, or whether to ask for more numbers to include in the average. As you may recall, the branch was dependent on the outcome of a specified condition, using the `IF ... THEN` statement. The HP-85 provides several forms of the `IF ... THEN` statement. One of them is:

`IF` numeric expression `THEN` statement number

The `IF ... THEN` statement makes a "decision" based upon the outcome of the numeric expression. If the expression is true, the `THEN` part of the statement is executed. If the outcome is false, execution continues with the statement following the `IF ... THEN` statement.

For example, suppose an accountant wishes to write a program that will calculate and print the amount of tax to be paid by a number of persons. For those with incomes of $10,000 per year or less, the amount of tax is 17.5%. For those with incomes of over $10,000, the tax is 20%. A flowchart for the program might look like this:



The diamond in the flowchart would be represented by an `IF ... THEN` statement in a BASIC program. Thus a sample solution to the problem might be:

```
10 DISP "INCOME";
20 INPUT I
30 IF I>10000 THEN 60
40 PRINT "TAX=";I*.175
50 GOTO 70
60 PRINT "TAX=";I*.2
70 END
```

''DO IF TRUE'' rule:
IF TRUE THEN EXECUTE.
IF FALSE THEN GOTO NEXT STATEMENT IN PROGRAM.

As you can see, we used a relational operation in the `IF ... THEN` statement. The `IF ... THEN` statement is most often used with relational operators (`=`, `<`, `>`, `<=`, `>=`, `<>` or `#`), although the decision can be based on the value of any numeric expression as we shall see later.

If the condition is true, i.e., if the income is greater than $10,000, then program control is transferred to statement 60. If the condition is false—in this case, if the income is less than or equal to $10,000—then the rest of the `IF` statement is ignored and the program continues at statement 40.

Now, test your program with values of $20,000 and $9,000. We ran the sample program below in print all mode by executing the `PRINT ALL` command to print all inputs and outputs.

RUN

```
INCOME?
20000
TAX= 4000
```

Computed 20% of income.

RUN

```
INCOME?
9000
TAX= 1575
```

Computed 17.5% of income.

Remember from our discussion of the logical evaluation (page 53) that an operation is assigned the value of 1 if it is true and a value of 0 if it is false. Thus, in an `IF... THEN` statement, if the outcome of a numeric expression has a value other than 0, it is considered true; if it has a value of 0, it is considered false.

Example: Write a program to compute $^1/x$. Since division by zero yields an error, use an `IF... THEN` statement to check for a zero input. Then load the program and run it for values of 0 and 9.

Here is a sample solution to the problem:

```
10 REM *RECIPROCAL*
20 DISP "ENTER NUMBER"
30 INPUT X
40 IF X THEN 80
50 DISP "THE RECIPROCAL OF ZERO"
60 DISP "IS UNDEFINED!!!"
70 GOTO 20
80 PRINT "1 /";X;"=";1/X
90 END
```

If X is any number other than 0, then the program branches to statement 80. The statement means the same as `IF X#0 THEN 80`. If X is 0, then execution continues with statement 50.

Before we ran this program, we executed the `PRINT ALL` command to print all inputs and outputs.

RUN

```
ENTER NUMBER
?
0
THE RECIPROCAL OF ZERO
IS UNDEFINED!!!
ENTER NUMBER
?
9
1 / 9 = .111111111111
```

Another form of the `IF... THEN` statement provides conditional execution of a statement without necessarily branching:

`IF` numeric expression `THEN` executable statement

Again, when the condition is true (or the value of the numeric expression is other than zero), the statement is executed. When the condition is false (the value of the numeric expression is zero), execution continues with the following statement.

All *executable* BASIC statements are allowed to follow `THEN` except the `FOR`, `NEXT`, and `IF` statements.

The following statements are not allowed after `THEN` because they are *declaratory*, not executable statements:

```
COM          IMAGE
DATA         INTEGER
DEF FN       OPTION BASE
DIM          REAL
FN END       SHORT
```

Example: Write a program to make Celsius/Fahrenheit temperature conversions such that:

1. If you enter a C, the temperature is converted from Celsius degrees to Fahrenheit according to the formula $F = 32 + 9/5 * C$.
2. If you enter an F, the temperature is converted from Fahrenheit to Celsius according to the formula $C = (F - 32) * 5/9$.
3. If neither C nor F is entered, nothing is printed.

If you wrote programs for problems 5.1.a. and 5.1.b., combine them. Use the second form of the `IF`... `THEN` statement in your program to determine which conversion is to be made.

Here is a listing of a sample solution:

```
 10 ! *TEMPERATURE CONVERSIONS*
 20 DISP "ENTER TEMPERATURE,F OR
 "
 30 DISP "ENTER TEMPERATURE,C"
 40 INPUT T,D$
•50 IF D$="F" THEN PRINT (T-32)*
 5/9;"C IS";T;"F"
•60 IF D$="C" THEN PRINT T;"C IS
 ";32+9/5*T;"F"
 70 GOTO 20
 80 END
```

Run the program to convert 0°C and 100°C to degrees Fahrenheit; 50°F and 98.6°F to degrees Celsius.

Here are the results printed from our program:

```
0 C IS 32 F
100 C IS 212 F
10 C IS 50 F
37 C IS 98.6 F
```

The ELSE Option

There's still more to the `IF`... `THEN` statement: `ELSE`. In the previous examples, if the numeric expression was evaluated as false, program execution continued with the next sequential statement following the `IF`... `THEN` statement. But if you specify the `ELSE` option with the `IF`... `THEN` statement, the program will instead perform the indicated `ELSE` instructions. This gives you tremendous power with conditional branching; six different forms of the `IF`... `THEN` statement are available.

`IF` numeric expression `THEN` statement number [`ELSE` statement number
or or
executable statement executable statement]

If the numeric expression is false and `ELSE` is specified, execution is transferred to the statement number following `ELSE` or the indicated `ELSE` statement is executed.

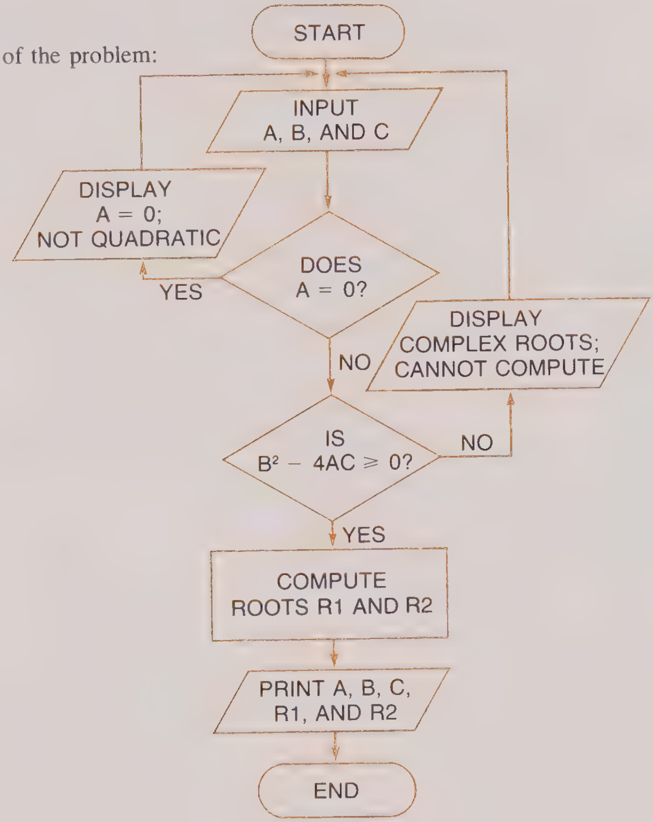
Let's look at an example.

Example: A quadratic equation is of the form $0 = ax^2 + bx + c$. If $a \neq 0$, its two roots may be found by the formulas

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Write a program to compute the roots of a quadratic equation given the values of the coefficients *a*, *b*, and *c*. If *a* is zero, display an error message and reenter new values. If $b^2 - 4ac$ is less than zero, then the square root of that value would give a warning message or an error. So make sure that $b^2 - 4ac$ is greater than or equal to zero before you compute the roots.

Here's a flowchart of the problem:


START
INPUT A, B, AND C
DOES A = 0?
YES
DISPLAY A = 0; NOT QUADRATIC
NO
IS B² – 4AC ≥ 0?
NO
DISPLAY COMPLEX ROOTS; CANNOT COMPUTE
YES
COMPUTE ROOTS R1 AND R2
PRINT A, B, C, R1, AND R2
END

In this sample solution we use two forms of the `IF`... `THEN`... `ELSE` statement. Study it carefully, then load the program and run it.

```
10 REM *ROOTS*
20 DISP "IF A QUADRATIC"
30 DISP "EQUATION IS OF THE"
40 DISP "FORM 0=A*X^2+B*X+C"
50 DISP "ENTER A,B,C"
60 INPUT A,B,C
70 D=B*B-4*A*C
80 IF A=0 THEN DISP "A=0.NOT QU
   ADRATIC.REENTER VALUES" ELSE
    100
90 GOTO 60
100 IF D>=0 THEN 120 ELSE DISP "
   COMPLEX ROOTS:  CANNOT COMPU
   TE  REENTER VALUES."
110 GOTO 60
120 R1=(-B+SQR(D))/(2*A)
130 R2=(-B-SQR(D))/(2*A)
140 PRINT "COEFFICIENTS:",A,B,C
150 PRINT "ROOTS:",R1,R2
160 END
```

If A=0, displays message, then continues to next statement. If A≠0, reads `ELSE 100` and program branches to statement 100.
If D>=0, branches to statement 120.
If D<0, displays `ELSE` message then continues to statement 110.

The instruction following `ELSE` in an `IF` ... `THEN` statement may be a statement number or an executable statement. Again, the same stipulations hold for `ELSE` as `THEN`; you may use any executable statement except `FOR`, `NEXT`, and `IF` and you may not use declaratory statements.

Run the program to find the roots of the equation $x^2 + x - 6 = 0$. Then run the program again to test the decisions with $x + 1 = 0$ and $x^2 + 2x + 2 = 0$; finally the roots of $3x^2 + 2x - 1 = 0$.

```
PRINTALL
RUN

IF A QUADRATIC
EQUATION IS OF THE
FORM 0=A*X^2+B*X+C
ENTER A,B,C
?
1,1,-6                                  Coefficients of x² + x − 6 = 0.
COEFFICIENTS= 1  1 -6
ROOTS= 2  -3                            Result.
```

```
RUN
IF A QUADRATIC
EQUATION IS OF THE
FORM 0=A*X^2+B*X+C
ENTER A,B,C
?
0,1,1                                   Coefficients of x + 1 = 0.
A=0;NOT QUADRATIC.REENTER VALUES        Displays message.
?                                       Asks for new input.
1,2,2                                   Coefficients of x² + 2x + 2 = 0.
COMPLEX ROOTS:  CANNOT COMPUTE.         Displays message.
REENTER VALUES.
?                                       Asks for new values.
3,2,-1                                  Coefficients of 3x² + 2x − 1 = 0.
COEFFICIENTS= 3 2 -1
ROOTS= .333333333333 -1                 Result.
```

For a more efficient and accurate method of finding the roots of a quadratic equation, refer to the Polynomial Evaluation program in your HP-85 Standard Pac.

The Computed GOTO Statement

There is one more form of *unconditional* branching that you should be aware of: the `ON` ... `GOTO` or computed `GOTO` statement.

`ON` numeric expression `GOTO` statement number list

The `ON` ... `GOTO` statement enables you to transfer program control to one of one or more statements, depending on the value of a numeric expression.

The numeric expression is evaluated and rounded to an integer. A value of 1 causes control to be transferred to the first statement specified in the statement list; a value of 2 causes control to be transferred to the second statement specified in the list, and so on. A value less than 1 causes an error. A value greater than the number of statements in the list also causes an error.

Essentially, the `ON... GOTO` statement is a combination of the `IF` statement and the `GOTO` statement.

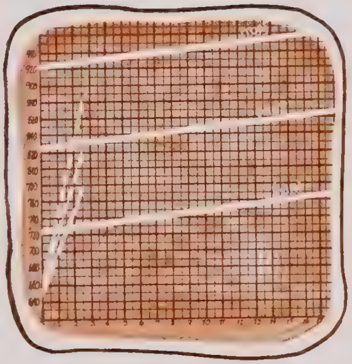
For example:

```
20 ON R GOTO 25, 80, 150
```

This statement says: if R=1, go to statement 25, if R=2, go to statement 80, and if R=3, go to 150. But if R<1 or if R>3, an error would occur.

Look at the following application of an `ON... GOTO` statement:

Example: The payroll clerk of a small firm wishes to write a program to compute the weekly wages of the employees according to the following payscales:



Payscale	Hourly Wage
1	$4.75
2	$5.50
3	$6.25

Also, overtime must be taken into account. If the employee works more than 40 hours in the week, the remaining hours should be multiplied by 1.5.

Sample Solution:

```
  10 PRINT "NAME"
  20 PRINT "HOURS","WAGES"
  30 PRINT
  40 E=0
  50 DISP "LAST NAME,FIRST INIT."
  60 INPUT N$
  70 DISP "HOURS WORKED";
  80 INPUT H
  90 IF H>40 THEN E=H-40
 100 H1=H-E+E*1.5
 110 DISP "PAY SCALE 1,2,OR 3";
 120 INPUT P
•130 ON P GOTO 140,160,180
 140 W=4.75*H1
 150 GOTO 190
 160 W=5.5*H1
 170 GOTO 190
 180 W=6.25*H1
 190 PRINT N$
 200 PRINT H,W
 210 PAUSE
 220 GOTO 40
 230 END
```

Since we have not dimensioned N$, the name can be no longer than 18 characters. (We'll discuss this later.)

Computes overtime.

Computed `GOTO` branches to 140 if P=1, 160 if P=2, and 180 if P=3.

Computes wages according to desired pay scale.

`PAUSE` after wages are printed for each employee. When the program is running, press CONT to continue.

Run the program for the following list of employees. Remember that if a comma is part of your string input, the string expression must be enclosed within quotes (e.g., enter `"JONES, J."` for the first name).

Name	**Hours**	**Payscale**
Jones, J.	43	2
Smith, K.	52	3
Fender, L.	40	1
Morris, D.	44	2

Your printout should look like this:

```
NAME
HOURS                    WAGES

JONES,J.
  43                     244.75
SMITH,K.
  52                     362.5
FENDER,L.
  40                     190
MORRIS,D.
  44                     253
```

Note: If the value of the numeric expression is less than one or greater than the number of statement numbers in the list, `Error 11` (argument out of range) occurs.

In the following example when statement 20 is executed for the third time, the value of I exceeds the number of statement numbers in the list.

```
10 I=1
20 ON I GOTO 30,30,60
30 DISP "I=";I
40 I=I*2
50 GOTO 20
60 DISP "I IS GREATER THAN 3"
70 END
```

Running the program:

RUN

```
I=1
I=2
Error 11 on line 20 : ARG OUT OF
 RANGE
```

FOR-NEXT Loops

Repeatedly executing a series of statements is known as looping. We have seen several loops in programs; the future value program contained a loop—as did the checkbook balancing program.

A clear and efficient way to create loops is to use the `FOR` and `NEXT` statements. The `FOR` and `NEXT` statements are used to enclose a series of statements, enabling you to repeat those statements a specified number of times.

`FOR` loop counter = initial value `TO` final value [`STEP` increment value]

⋮

`NEXT` loop counter

The `FOR` statement defines the beginning of the loop and specifies the number of times the loop is to be executed. The loop counter must be a simple numeric variable.

The initial, final, and increment values can be any numeric expression. If the increment value is not specified, the default value is one.

`FOR-NEXT`
loop range

```
 50 FOR I=1 TO 5
 60 PRINT I
 70 NEXT I
 80 PRINT "FINISHED WITH LOOP;"
 90 PRINT "I NOW EQUALS";I
100 STOP

RUN
  1
  2
  3
  4
  5
FINISHED WITH LOOP;
I NOW EQUALS 6
```

The `FOR-NEXT` loop will be executed five times: when I=1,2,3,4, and 5. Each time the `NEXT` statement is executed, the value of I is incremented by 1. But when the value of I passes the final value, that is, when I=6, the loop is finished, and execution continues with the statement following the `NEXT` statement (in this case, 80).

The `FOR` statement does three things for your program:

1. It sets the loop counter to the initial value.
2. It tells the computer what the final value for the index may be (and thereby when to stop looping).
3. It tests to see if the counter has gone beyond the final value. If so, the program exits the loop; if not, the program continues with the statement following the `FOR`.

The `NEXT` statement does two things for your program:

1. It increments the loop counter.
2. It defines the end of the loop. After the program has completed the loop the number of times specified, execution continues with the statement following `NEXT` (if, of course, you have not branched elsewhere within the loop).

Example: Use a `FOR-NEXT` loop to compute and print the area of a circle with an integer radius from 15 centimeters to 20 centimeters, according to the formula $A = \pi r^2$.

```
AUTO
10 FOR R=15 TO 20
20 A=PI*R*R
30 PRINT "RADIUS=";R;"AREA=";A
40 NEXT R
50 STOP
```

Notice that the initial value does not have to be 1.

Again, we set the initial value and the final value with the `FOR` statement. When R exceeds 20, program execution is transferred to the statement following the `NEXT` statement.

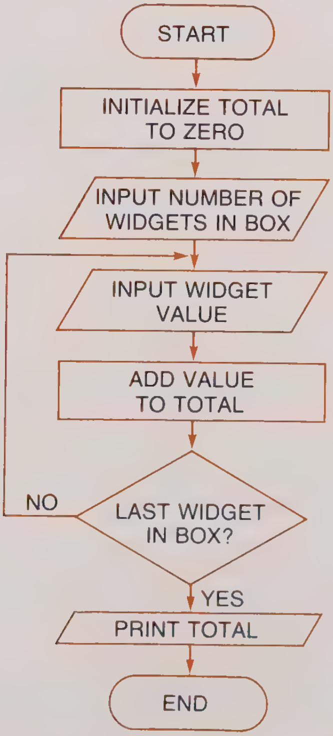
```
RUN
RADIUS= 15 AREA= 706.858347059
RADIUS= 16 AREA= 804.247719318
RADIUS= 17 AREA= 907.920276887
RADIUS= 18 AREA= 1017.87601976
RADIUS= 19 AREA= 1134.11494795
RADIUS= 20 AREA= 1256.63706144
```

The loop is executed 6 times from R=15 through R=20.

You can also use variables or numeric expressions to specify the initial or final values.

Example: Suppose you are a widget maker. The shipping department in the widget factory can pack widgets in a variety of ways—rarely do two boxes contain the same number of widgets. Since widgets come in various shapes and sizes, the value of each widget varies. But you want to insure the box for the true value of the widgets inside. Write a program to accept the number of widgets in a particular box and then accept the value of each widget in the box, compute the total, and print the value to be insured.

Your flowchart might look like this:


START
INITIALIZE TOTAL TO ZERO
INPUT NUMBER OF WIDGETS IN BOX
INPUT WIDGET VALUE
ADD VALUE TO TOTAL
LAST WIDGET IN BOX?
NO
YES
PRINT TOTAL
END

Sample Program:

```
   10 REM *WIDGETS*
   20 T=0
   30 DISP "ENTER NUMBER OF WIDGET
      S"
   40 INPUT N
●  50 FOR I=1 TO N
   60 DISP "WIDGET VALUE";
   70 INPUT W
   80 T=T+W
●  90 NEXT I
  100 DISP "TOTAL VALUE OF BOX=";T
  110 END
```

Here, you actually input the final value of the loop.
`FOR-NEXT` loop range.

Now run the program for a box of five widgets, with individual values of $3.50, $4.95, $2.60, $18.50, and $5.10:

```
RUN
ENTER NUMBER OF WIDGETS
?
5
WIDGET VALUE?
3.50
WIDGET VALUE?
4.95
WIDGET VALUE?
2.60
WIDGET VALUE?
18.50
WIDGET VALUE?
5.10
TOTAL VALUE OF BOX= 34.65
```

As we mentioned earlier, it is possible to use expressions in the `FOR` statement as either the initial value or the terminating value. For example, you could have a problem that requires you to have statements like the following:

```
130 FOR I=N/2 TO N*2-1
```
or
```
266 FOR J=1 TO N*8
```
or
```
470 FOR K=2*J TO 1000
```

In these instances, when the `FOR` statement is executed, the first expression is evaluated and the *numeric value* is stored as the initial value; then the second expression is evaluated and that *value* is stored as the terminating value. After the initial and terminating values have been stored, the variable N can be changed without altering either of them. But the loop counter variable should not be changed within a loop by an assignment statement.

For example, this program has problems:

```
100 FOR I=1 TO 10
110 I=3
120 NEXT I
130 END
```

This program would create an infinite loop, like those we've seen before, except worse since nothing is displayed or printed. The variable I is reset below the final value each time the program executes the loop.

Changing the Increment Value

In all of the `FOR-NEXT` loops above, the computer increments the counter by 1 each time through the loop. But you are not limited to just 1. You can use any stepping value, positive, negative, or non-integer, with the `STEP` parameter.

For example, suppose you wish to print the odd integers from 1 to 10. You could use a `FOR-NEXT-STEP` loop like this:

```
10 FOR I=1 TO 10 STEP 2
20 PRINT I;
30 NEXT I
40 PRINT
50 END
```

```
RUN

 1 3 5 7 9
```

The initial value of I is 1. Each time `NEXT I` is executed, I is incremented by 2. In this program, I = 1, 3, 5, 7, and 9. When I reaches 11, the loop is exited and the program ends. Since the `PRINT` statement in line 20 ends with a semicolon, an extra `PRINT` statement completes the print message and outputs it to the printer.

You can also decrement the loop counter.

Example: Write a program that requests a number and computes its factorial. A factorial is an integer multiplied by all of the other integers below it (down to 1). For instance, 6! = 6 × 5 × 4 × 3 × 2 × 1. (Also consider limiting the size of the number a user may give. What happens if a negative number or a noninteger is entered?)

Enter the following program into the system:

```
 10 REM *FACTORIAL*
 20 DISP "FACTORIAL OF";
 30 INPUT N
 40 IF FP(N)=0 AND N>=0 THEN 70
 50 DISP "POSITIVE INTEGERS ONLY"
 60 GOTO 20
 70 F=1
 80 FOR P=N TO 1 STEP -1
 90 LET F=F*P
100 NEXT P
110 DISP "FACTORIAL=";F
120 END
```

Check to make sure the number is a positive integer.

Loop counter is decremented from N to N−1, and so on.

Now run the program to find the factorials of 4 and 24.

```
RUN
FACTORIAL OF?
4
FACTORIAL= 24
```

Result.

```
RUN
FACTORIAL OF?
24
FACTORIAL= 6.20448401736E23
```

Result. Remember, the computer overflows with numbers larger than $9.99999999999 \times 10^{499}$.

Let's see how the factorial of 4 was computed. After you input 4, the initial value of the FOR statement was set to 4. So the program read the statement as:

```
FOR P=4 TO 1 STEP -1
```

The values for F were computed as follows:

`F=1*4`	First time through loop.
`F=4*3`	Second time through loop; P=4−1.
`F=12*2`	Third time through loop; P=3−1.
`F=24*1`	Last time through loop; P=2−1.

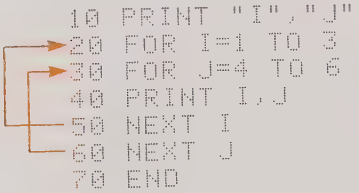
When the `NEXT` statement decrements the P value to 0, the loop is exited.

Nested Loops

When one loop is contained entirely within another, the inner loop is said to be nested. A loop can be contained within a loop that is contained within a loop ... (up to 255 nested loops), as long as the loops do not overlap each other.

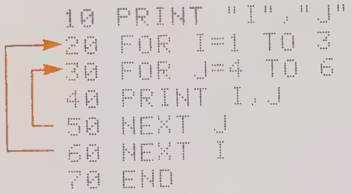
A `FOR-NEXT` loop cannot overlap another `FOR-NEXT` loop, for instance:

Incorrect Nesting



```
10 PRINT "I","J"
20 FOR I=1 TO 3
30 FOR J=4 TO 6
40 PRINT I,J
50 NEXT I
60 NEXT J
70 END
```

Correct Nesting



```
10 PRINT "I","J"
20 FOR I=1 TO 3
30 FOR J=4 TO 6
40 PRINT I,J
50 NEXT J
60 NEXT I
70 END
```

In the incorrect nesting example, the I loop is activated and then the J loop is activated. But the J loop is cancelled when `NEXT I` is executed because it's an inner loop. When the I loop is completed and `NEXT J` is accessed, `Error 47 on line 60` is displayed. This is because the J loop was cancelled and was not reactivated after the last I loop.

Run the correct nesting example now to view the looping process:

```
RUN
    I     J
    1     4
    1     5
    1     6
    2     4
    2     5
    2     6
    3     4
    3     5
    3     6
```

The J loop is completed before I is incremented.

Now I=2 and the program runs through the J loop again.

Finally I=3; the J loop is executed once again. When I reaches a value of 4, the program halts.

FOR-NEXT Loop Considerations

- Execution of a `FOR-NEXT` loop should always begin with the `FOR` statement. Branching into the middle of a loop (with statements like `GOTO` or `IF`) will produce `Error 47` if the `NEXT` statement is executed before the program executed the corresponding `FOR` statement.
- Execution of a loop normally ends with a `NEXT` statement. It is permissible to transfer program control out of the loop by a statement within the loop. After an exit is made through a branch within the loop, the current value of the counter is retained and is available for later use in the program. In this case, it is permissible to reenter the loop either at a statement within the loop, or at the `FOR` statement (thereby reinitializing the counter).
- A `FOR-NEXT` loop will *not* be executed if the initial value is greater than the final value when a positive `STEP` value is used, or if the initial value is less than the final value when a negative `STEP` value is used.
- An often overlooked aspect of `FOR-NEXT` looping is that the actual value of the counter when the loop is complete does not equal the final value. The `NEXT` statement increments or decrements the loop counter *past* the final value before the loop is exited. (We'll see an example of this in the graphics section, Padding the `FOR-NEXT` loop.)

Problems

7.1 As an avid sports fan, you decide to write a program that will help you keep score during an important basketball game between the Aakerville Aardvarks and the Wiggenberg Wombats. You can enter an *A* or *W* to signify a field goal (worth 2 points) for the appropriate team, and an *a* or *w* to signify a free throw (worth 1 point). The score should be printed after each entry.

7.2 The common game of "Buzz" offers a challenge to a person's number skills. This version, called "Beep," requires you to program the HP-85 to successfully complete the same game. The game consists of counting (displaying) numbers from 1 to 100. However, for any number that is evenly divisible by 7 or contains a 7, the display should leave a blank and the HP-85 should "beep." If the number both contains a 7 and is evenly divisible by 7, two "beeps" should be sounded.

Hint: The "ones" digit of a two-digit integer can be found as `10*FP(X/10)` or `XMOD10`.

7.3 Here is a check to see whether you and the HP-85 can communicate using "mental" telepathy. Write a program that uses the `RND` random number generator to "pick" a number from 1 to 5, waits for 5 seconds while "concentrating" on the number, and then requests from you the number that comes to your mind. The display should indicate whether your entry is correct or incorrect. After every 10 picks, the printer should list a summary of your accuracy and indicate whether it is better or worse than that expected by chance (20% accuracy). The random "picks" can be generated by `P=IP(1+5*RND)`.

7.4 Boy Scout Jeffrey Goodfellow is preparing for his compass-course test, in which he must follow several legs of a course and attempt to be within an allowable error of the finish point. Each leg of the course is defined by a magnetic bearing (θ) to be followed and the distance (d) to be traveled. Jeffrey realizes that each leg can be converted to northerly and easterly distances (d_n and d_e) according to:

$$d_n = d \times \cos(\theta)$$

$$d_e = d \times \sin(\theta)$$

If the northerly and easterly distances are summed for all the legs in the course, these two sums can be used to determine the direct bearing (θ_f) and distance (d_f) of the finish point relative to the starting point:

$$\theta_f = \arctan\,(d_e/d_n)$$

$$d_f = \sqrt{d_e^2 + d_n^2}$$

If he had a program to perform these calculations, Jeffrey could check his accuracy during his practice sessions. Write a program that requests the bearing and distance for each leg of the compass course. (A distance of zero should indicate that all of the legs have been entered.) The program should produce a listing of the bearings and distances, and then give the direct bearing and distance of the finish point relative to the starting point. Use the `ATN2(Y,X)` function to compute θ_f so that the proper angle is chosen. (If θ_f is negative, add 360° or use θ_fMOD360 to obtain the bearing in the correct form.)

Hint: Don't forget the `DEG` statement.

7.5 Your medical supplies business has offices in Britain, France, and the United States. With such an arrangement, you must frequently convert monetary values among the three currency systems: British pound, French franc, and U.S. dollar. In order to facilitate these conversions, you decide to write a program to compute them for you. Each currency system is to be denoted by a code number. The program is to be initialized each day by entering equivalent monetary values in each currency. Each required conversion should begin by entering the currency code and amount to be converted; the currency system and equivalent amount is to be printed for each system. On a certain morning, 1 British pound is equivalent to 8.3981 French francs and 1.8248 U.S. dollars. At these rates, find the equivalent values of a patient lift worth 284 British pounds and a hospital bed valued at 1205 U.S. dollars.

7.6 The mayor of Dimsburg has directed Elmo Rumple, the town statistician, to study the problem of motorists having to stop at all three of Dimsburg's traffic lights. Elmo confines his analysis to those motorists who are delayed at all three lights. He assumes that each car arrives randomly at each red light, indicating that the delay at each light is uniformly distributed between 0 and 1 minute (the duration of a red signal). The total delay in Dimsburg is therefore the sum of the three uniformly-distributed delays. Elmo wants to compute the probability that this delay is shorter than various time intervals. From his vast experience, he knows that this probability is given by the following function (called a distribution function).

$$\text{Prob(delay} < \text{T)} = \begin{array}{ll} 0 & \text{for T} < 0 \\ \text{T}\wedge 3/6 & \text{for } 0 \leqslant \text{T} < 1 \\ .5-\text{T}*(1.5-\text{T}*(1.5-\text{T}/3)) & \text{for } 1 \leqslant \text{T} < 2 \\ -3.5+\text{T}*(4.5-\text{T}*(1.5-\text{T}/6)) & \text{for } 2 \leqslant \text{T} < 3 \\ 1 & \text{for T} \geqslant 3 \end{array}$$

Help Elmo by writing a program to compute the probability of a total delay that is less than any specified time. (Use an `ON... GOTO` statement to branch to the proper equation.)

Section 8

Using Variables: Arrays and Strings

As we mentioned earlier, there are three types of numeric variables available with the HP-85: `REAL` (full precision), `SHORT`, and `INTEGER` numbers. A fourth type of variable deals with character strings. Numeric variables can have two forms: simple (non-subscripted) and array (subscripted). Strings may also be subscripted, but not in the form of an array.

In this section, we discuss array and string variables, their functions, and how to use them.

Array Concepts

An array variable (or simply, an array) is a collection of data items of the same type under one name. An array may have one or two dimensions. For instance, a one-dimensional array (often called a vector) might be thought of as a *list* of items; there may be several rows but only one column. A two-dimensional array (often called a matrix) is like a *table* of values; there may be several rows and several columns of items.

Suppose we have the following list of numbers:

1
4
9
16
25

We could store this list of numbers, in the order shown, in a one-dimensional array.

If we name this set of numbers array S, we can specify the individual elements of S by using subscripts.

If numbering of array subscripts begins with 0, the elements of array S are specified as:

S(0) =1
S(1) =4
S(2) =9
S(3) =16
S(4) =25

Subscripts	Array Elements
0	1
1	4
2	9
3	16
4	25

If the numbering of array S begins with 1, then the elements of array S are:

S(1) =1
S(2) =4
S(3) =9
S(4) =16
S(5) =25

Subscripts	Array Elements
1	1
2	4
3	9
4	16
5	25

We need to use a two-dimensional array to store the values in the following table:

Number	Square	Square root	Factorial
1	1	1	1
2	4	1.41421356237	2
3	9	1.73205080757	6

This table contains 3 rows and 4 columns for a total of 12 values.

If subscript numbering begins at 0, the elements are identified as follows:

```
D(0,0) =1        D(0,1) =1        D(0,2) =1                 D(0,3) =1
D(1,0) =2        D(1,1) =4        D(1,2) =1.41421356237     D(1,3) =2
D(2,0) =3        D(2,1) =9        D(2,2) =1.73205080757     D(2,3) =6
```

Array D

Subscript	0	1	2	3
0	1	1	1	1
1	2	4	1.41421356237	2
2	3	9	1.73205080757	6

Each element in array D is specified by its location in the array with two subscripts, separated by a comma, and enclosed within parentheses. The first subscript designates the ''row'' in the array; the second subscript designates the ''column.''

If numbering of the subscripts begins with 1, array D would be represented:

Subscript	1	2	3	4
1	1	1	1	1
2	2	4	1.41421356237	2
3	3	9	1.73205080757	6

Thus, 9 would be represented as D(3,2); 6 would be represented as D(3,4).

Array names are the same as simple variable names; an array name may be a letter from A through Z, or a letter immediately followed by a digit from 0 through 9. But whenever an array is specified, it must be followed by subscripts enclosed within parentheses, otherwise it specifies a simple variable. The range of each subscript is an integer from 0 through 32767, but the maximum array size is determined by available memory. A non-integer subscript is rounded to the nearest integer if it is within the dimensioned range of subscripts.

Arrays are extremely convenient for handling large groups of data within a program because a group of different values are known under the same name. The different values (or *elements* of the array) are distinguished in name by subscripts to the array name.

An array name followed by a single subscript enclosed within parentheses specifies a one-dimensional array or an element of that array. An array name followed by two subscripts separated by a comma, both enclosed within parentheses, specifies a two-dimensional array or an element of that array. (No more than two subscripts are allowed.) Whether the array name is understood as the whole array or as a specific element depends on the type of statement that is used. Declaration statements refer to the whole array, executable statements usually refer to an array element.

Declaring and Dimensioning Variables

Five variable declarative statements are available to dimension arrays and strings and declare the precision of numeric variables:

```
COM
DIM
INTEGER
SHORT
REAL
```

They can be anywhere in a program (except that in multistatement lines, they must be the last statement in the line). The subscripts of a variable must specify the physical or maximum size of the variable.

Lower Bounds of Arrays

Earlier we saw that subscript numbering can begin with 0 or 1. The HP-85 assumes that all array subscripts begin at 0 unless you specify otherwise with an `OPTION BASE` statement.

When dimensioning arrays, you may want to specify that the lowest numbered subscript be 1 rather than 0.

```
OPTION BASE 1
```

This statement must come before any array variables are referenced in a program. `OPTION BASE 1` tells the computer to begin numbering all subscripts of arrays with 1.

The real advantage to using `OPTION BASE 1` is that you can refer to an array element directly by its position in the array without wasting element 0. Thus, the first element in a one-dimensional array S is S(1) rather than S(0); the second element is S(2) rather S(1) and so on. And if array S contained 10 elements, it would be declared as S(10) rather than S(9).

If `OPTION BASE 1` is not declared in a program, you may wish to include the statement `OPTION BASE 0` for documentation purposes. But this is not necessary since `OPTION BASE 0` is the *default* array counting system at power on. There may only be *one* `OPTION BASE` statement in a program.

The `OPTION BASE` statement cannot be executed from the keyboard.

The DIM Statement

The `DIM` *(dimension)* statement is used to dimension (allocate memory) and reserve memory for full precision numeric arrays. It is also used to dimension and reserve storage space for strings.

`DIM` item [, item...]

The item can be:

- A numeric array, with subscripts enclosed within parentheses.
- A string, with the number of characters enclosed within brackets.

The `DIM` statement specifies the upper bound of an array and the maximum number of characters that a character string may have.

Remember that the HP-85 assumes that the lower bound of an array is 0 unless you specify it to be 1 with `OPTION BASE`.

Examples:

`10 DIM A(100)`	Declares a one-dimensional array A of 101 elements; A(0),..,A(100).
`20 DIM B(3,2),C$[56]`	Declares a two-dimensional array B of 12 elements (4 by 3) and a character string C$ of 56 characters maximum. (Refer to our discussion of strings on pages 51 and 52.)

With `OPTION BASE 0` the number of elements in each dimension of a numeric array is calculated by adding one to each upper bound subscript. Then the resulting values are multiplied together to yield the total number of elements in a two-dimensional array.

Examples:

`5 OPTION BASE 1`	Declares the lower bound of all arrays to be 1.
`10 DIM A(100),B(3,2),C$[56]`	Dimensions an array A with 100 elements array B with 6 elements (3 by 2), and a string C$ with 56 characters.

The memory allocated to a character string is not affected by `OPTION BASE`. In a `DIM` statement, the number within brackets always refers to the number of characters allocated to the string. The maximum number of characters that may be specified for a string is 32767, but this is limited by available memory.

Type Declaration Statements

All numeric variables (simple and array) are assumed to be full precision variables (type `REAL`), unless they appear in a type declaration statement. A type declaration statement specifies the type of variable, `REAL`, `SHORT`, or `INTEGER`.

`INTEGER` numeric variable$_1$ [`(`subscripts `)`] [`,` numeric variable$_2$ [(subscripts)]...]
`SHORT` numeric variable$_1$ [`(`subscripts `)`] [`,` numeric variable$_2$ [(subscripts)]...]
`REAL` numeric variable$_1$ [`(`subscripts `)`] [`,` numeric variable$_2$ [(subscripts)]...]

The `INTEGER` statement dimensions and reserves memory for integer precision variables—simple and array.

The `SHORT` statement dimensions and reserves memory for short precision variables—simple and array.

And the `REAL` statement dimensions and reserves memory for full precision variables—simple and array.

Since the `DIM` statement is used to dimension full-precision variables, and undeclared simple variables are assumed to be full-precision, the `REAL` statement is only useful for documentation purposes.

Examples:

`10 INTEGER A,B,C(10)`	Declares variables A and B to be integers; declares and dimensions array C to 11 integer elements, assuming OPTION BASE 0.
`20 SHORT P(20,25),P1,P2`	Declares and dimensions short-precision elements for array P; declares variables P1 and P2 to be short-precision.
`30 REAL X5,D(4,4)`	Declares array D and variable X5 to be type REAL.

The COM Statement

The `COM` *(common)* statement is used to dimension and reserve variables to be held in common in two or more programs. `COM` is primarily used with the `CHAIN` statement (section 11) to pass variables between programs. A `COM` statement may also be used to deallocate a program before it is stored (refer to page 263). The variables in common must agree in type and size between programs that are `CHAIN`ed.

`COM` item [, item,...]

The item can be:

- A simple numeric variable.
- A subscripted array.
- A string with number of characters enclosed within brackets.

In addition, any one of the type words— `INTEGER`, `SHORT`, or `REAL`—may precede one or more variables.

Example:

```
25 COM A,B(4,3),C$[5],D,INTEGER
   E,F$[24],G,SHORT H(5), J
```

The variables A,B(4,3), and D are full precision. Full precision is assumed at the beginning of the `COM` list and for numeric variables declared after a type `REAL` declaration. From left to right in a given `COM` list, all variables following a numeric type word have that precision until another type word appears in the list. Thus, both H(5) and J are short precision.

`COM` statements in separate programs that are linked with the `CHAIN` statement must agree in number and type of variable. Variables held in common are reset to undefined values by executing `SCRATCH`, `RUN`, or `INIT`.

About Variable Declarations

- The `COM` statement must be used in a program, not from the keyboard, and may not appear within a function definition.
- The location in a program of `DIM`, `COM`, type declaration is arbitrary, though they must be after an `OPTION BASE` statement and before any other reference to the dimensioned variable.

- The `DIM` statement need not be used to assign memory space for strings with 18 characters or less or for arrays that have upper bounds of 10 or less. Thus, you do not need to use `DIM` with an array A(5,5) (of 25 or 36 elements depending on the lower bound) or a string C$="SQUARES". But array A(5,5) will be implicitly dimensioned to be A(10,10) and string C$ will be implicitly dimensioned to have 18 characters rather than 7 (the number of characters in "SQUARES"). Thus, you may wish to use `DIM` to conserve memory with small arrays and strings.
- A program can have more than one `DIM`, `COM`, or type declaration statement, but the same variable name can be declared only once in a program. Therefore, arrays of differing dimensions or variables of different types cannot have the same name. But the same name may be used for a simple numeric, a string, and a numeric array.

String Expressions

The simplest form of a string expression is text within quotes. This is called a *literal* string and can be made up of any characters excluding quotation marks.

For example, execute the first two statements:

```
C$= " STRING "
DISP C$;C$
 STRING  STRING
```

This string expression contains eight characters; two spaces and the word "STRING". Quotation marks are not included in a literal string because they mark the beginning and end of the string.

The forms that a string expression can take are:

- Text within quotes.
- String variable name.
- Substring.
- String concatenation operation (`&`).
- String function.
- Any logical combination of the above.

As with numeric expressions, a string expression can be enclosed in parentheses if necessary.

In this section, we discuss substrings and string functions.

Thus far, you have learned to assign a literal string to a string variable and to join two strings together using the ampersand (`&`) as the string concatenator (page 52).

You have also seen that unless the size of a string variable is specified in a `DIM` statement, it is implicitly dimensioned to be a maximum of 18 characters in length.

```
10 DIM A$[15], F$[28], H$[100]
```

The statement above dimensions string variable A$ to be a maximum of 15 characters, F$ to be a maximum of 28 characters, and H$ to be a maximum of 100 characters. Brackets (not parentheses) must surround the number of characters to be included in the string variable.

Substrings

A substring is a part of a string made up of zero or more contiguous characters. A substring is specified by placing subscripts in brackets after the string name. There are two forms a substring can have:

- String variable name [character position]

  The character position is a numeric expression which is rounded *(not truncated)* to an integer. The substring is made up of that character and all following it.

- String variable name [beginning character position , ending character position]

  This substring includes the beginning and ending characters and all in between. The character positions must be within the dimensioned number of characters. If the first subscript is exactly one greater than the second subscript, the null string (" ") is specified.

Example: Suppose we dimension and assign string A$ as follows:

```
DIM A$[25]
A$="A STRING OF 25 CHARACTERS"
```

Spaces and blanks are also characters.

Now look at the various examples of substrings of A$:

```
A$[5] = RING OF 25 CHARACTERS
A$[15] = CHARACTERS

A$[1,8] = A STRING
A$[13,14] = 25
A$[5,8] = RING
```

One subscript denotes a substring from that character position to the end of the main string.

Two subscripts denote a substring that includes the characters in the positions specified and all characters in between.

Modifying String Variables

There are a variety of ways that you can modify a string or substring by another string or substring. For instance, a part of a string can be changed or characters can be added or deleted. The modifying string can be any string expression.

The length and content of a modified string depend not only on the characteristics of the modifying string, but also on the number of subscripts given for the original string.

poly·nona·hexa·hepta
penta·deca·dodeca
GON
HEDRON

Replacing a String

You can replace the complete string of characters with another string using an assignment statement.

For example:

Press	Display	
A$ = "HELLO" END LINE	A$ = "HELLO"	Assigns string to A$.
B$ = "GOODBYE" END LINE	B$ = "GOODBYE"	Assigns string to B$.
B$ = A$ END LINE	B$ = A$	Assigns B$ the expression in A$.
B$ END LINE	B$	Recalls B$ to verify.
	HELLO	

As you can see, B$ was reassigned the string in A$. When no subscripts are specified for either variable, the string is completely replaced with the new string. You can also reassign string variables by typing the new string within quotes.

For example:

Press	Display	
A$ ="HI" END LINE	A$="HI"	A string variable contains the characters most recently assigned to it.
A$ END LINE	A$	
	HI	
A$ ="BYE" END LINE	A$="BYE"	
A$ END LINE	A$	Recalls A$ to verify.
	BYE	

Replacing Part of a String

After you have assigned a character string to a variable, you can replace one substring with another substring. The original string can be lengthened or shortened. But if you attempt to lengthen the string beyond its dimensioned length, you will cause an error.

Change substrings by specifying the subscripts of the characters to be changed and the new substring.

For example:

Press	Display	
H$ ="HAPPENING" END LINE	H$="HAPPENING"	Assigns substring to H$ beginning with character 7.
H$[7] ="STANCE" END LINE	H$[7]="STANCE"	
H$ END LINE	H$	
	HAPPENSTANCE	Lengthened string.
H$[5] ="ILY" END LINE	H$[5]="ILY"	Assigns substring to H$ beginning with character 5.
H$ END LINE	H$	
	HAPPILY	New string.

If characters added to a string are not contiguous (in other words, some character positions are left unassigned), blank spaces will fill the unassigned characters in the string.

For example:

```
W$="C."
W$[5]="JACKSON"
W$
C.  JACKSON
```

Since the third and fourth character positions of W$ have not been assigned characters, they are filled with blank spaces.

You can also replace the beginning or the middle of a string with another substring. Do this by using two subscripts to specify the first and last character positions of the substring to be replaced.

If the new substring is shorter than the substring that you replace, the remainder of the new substring is replaced with blanks; if the new substring is longer than the one you replace, the remainder of the new substring is truncated.

For example:

Press	**Display**	
Z$ ="HEPTAGON" END LINE	`Z$ = "HEPTAGON"`	
Z$[1,3] = "PEN" END LINE	`Z$[1,3] = "PEN"`	Replaces characters 1 through 3 of Z$ with specified substring.
Z$ END LINE	`Z$` `PENTAGON`	
Z$[1,4] = "HEX" END LINE	`Z$[1,4] = "HEX"`	Since you replaced characters 1 through 4 with a string of length 3, the fourth character is a blank.
Z$ END LINE	`Z$` `HEX AGON`	

Press	**Display**	
Z$[1,4]= "DODEC" END LINE	`Z$[1,4]="DODEC"`	If you try to replace four characters with five characters, the fifth character is truncated.
Z$ END LINE	`Z$` `DODEAGON`	

Another way to specify the null string is to make the first subscript one larger than the second subscript in a substring. Thus the following statements are equivalent:

```
N$=""
N$=A$[4,3]
N$=C$[8,7]
```

Each specifies no blanks, no characters. (A$ and C$ must have been previously assigned values or an error occurs.)

String Functions

The HP-85 provides seven different functions to enable you to determine the length of a string and analyze and manipulate its contents.

These functions are:

String Function (Parameter)	Meaning
`LEN(`string`)`	Length of string.
`POS(`string 1, string 2`)`	Position of string 2 in string 1.
`VAL(`string`)`	Returns the numeric value of a string expression composed of digits.
`VAL$(`numeric expression`)`	Generates a string representing the numeric value of a numeric expression.
`CHR$(`numeric expression`)`	Converts a numeric expression to the appropriate character.
`NUM(`string`)`	Returns the decimal value of the first character of the string.
`UPC$(`string`)`	Converts all lowercase letters in string to uppercase letters.

The Length Function

The `LEN` *(length)* function returns the number of characters in a string expression.

`LEN(`string expression`)`

The current length of a string expression is returned. Remember, a string variable isn't always "full"; the length isn't necessarily the maximum length that you give it in a `DIM` statement.

Examples:

`LEN("123")`	Length of string "123".
`3`	Result of `LEN` function: 3 characters long.
`A$="length*width"`	Assigns string to variable A$.
`LEN(A$)`	Finds length of A$.
`12`	Result: 12 characters long.
`LEN(A$[1,6])`	Finds length of first six character positions in A$.
`6`	Result: 6 characters long.

Notice that the string expression may be quoted text, a string variable name, or a substring. The expression must be enclosed within parentheses.

Example: Write a program that will let you enter a character string of up to 40 characters in length. Then, using the `LEN` function, compute and display the word with the characters in reverse order. For instance, if you input `CAT`, the program should display `TAC`.

saw-was
rats-star
stop-pots
lag-gal

```
10 DIM W$[40], R$[40]
20 R$=""

30 DISP "WORD";
40 INPUT W$
50 FOR I=LEN(W$) TO 1 STEP -1

60 R$=R$&W$[I,I]

70 NEXT I
80 DISP R$
90 END
```

Dimensions the string variables to be a maximum of 40 characters long. Initializes R$ to the null string.
Displays a message to prompt an input.
Inputs a word.
Uses length of word for loop counter and counts in reverse order.
With the string concatenator, adds characters to variable R$ in reverse order.
Defines end of `FOR-NEXT` loop.
Displays reversed word.

After you enter the program above, try spelling some words backwards!

RUN

```
WORD?
CAT
TAC
```

RUN

```
WORD?
YELLOW PAGES
SEGAP WOLLEY
```

RUN

```
WORD?
INTERNATIONAL
LANOITANRETNI
```

The program reverses the order of the characters in the string—including spaces between words.

The Position Function

The `POS` *(position)* function determines the position of a substring within a string.

`POS(`in string expression `,` of string expression `)`

If the second string is contained within the first, the `POS` function returns the position of the first character of the second string within the first string. If the second string is not contained within the first string, or if the second string is the null string, the value returned by the function is zero. If the second string occurs in more than one place within the first string, only the first occurrence is given by the function.

Examples:

```
POS("ABABC1234","123")
 6
```

Finds position of second string in first string. Result: second string begins at sixth character position.

```
POS("ABABC1234", "AB")
 1
```

Result: first occurrence of "AB" within first string.

```
A$="COMPOSER"
B$="POSE"
POS(A$,B$)
 4
```

Position of B$ in A$.
Result: B$ begins at fourth character position of A$.

Be sure to separate the string expressions by a comma.

Converting Strings to Numbers

Normally, the characters in a string are not recognized as numeric data and can't be used in numeric calculations. That's because you usually want the string to be quoted literally, character for character.

With the `VAL`*(value)* function the numeric value of a string or a substring of digits, including an exponent, can be used in calculations.

`VAL(`string expression`)`

For example, suppose

```
A$="ROMEO,J; 257684321"
```

If you want to obtain the numeric value rather than the literal substring of ‘‘257684321‘‘, you must use the `VAL` function:

```
VAL(A$[10])
 257684321

A=VAL(A$[10])
```

Gives *numeric value* of A$ from character 10 to end of string. This is a number, *not* a string. Note that the system indents positive numbers; i.e., the space before the number is for the sign (if any). Now, this number can be assigned to a numeric variable and it can be used in numeric calculations.

```
A$[10]
257684321
```

This is a *substring* of A$. A$[10] is not a numeric value. Notice that no space precedes the number to specify the sign. The string cannot be assigned to a numeric variable nor can it be used in numeric calculations.

When you use the `VAL` function, the first character in the string to be converted must be a digit, a plus or minus sign, a decimal point, or a space. A leading plus sign or space is ignored; a leading minus sign is taken into account. The remaining characters in the string or substring must be digits, a decimal point, or an `E`. An `E` character after a numeric and followed by digits (including sign) is interpreted as an exponent of 10.

Examples:

```
VAL("4E-2")
 .04

VAL("-1234.567")
-1234.567
```

The function outputs the number in standard format.

A string can contain more than one number. All contiguous numerics are considered a part of the number until a non-numeric is reached in the string.

Example:

```
B$="43 SCORE 59"
VAL(B$)
 43

VAL(B$[9])
 59
```

As long as the first character is a numeric, the `VAL` function converts the string to a number until it reaches a non-numeric character.

But you can convert the remaining numerics in the string by subscripting the string variable. Here we specify the numeric value of B$ from character position nine to the end of the string.

Converting Numbers to Strings

The `VAL$` function is nearly the inverse of the `VAL` function. With the `VAL$` function, you can convert a number to a string representation of the number in standard format.

`VAL$(`numeric expression`)`

Examples:

```
V$=VAL$(120)
V$
120
```

Result of executing V$; V$="120".

```
W$=VAL$(4*8)
W$
32
```

W$ = "32".

```
X$=VAL$(SQR(64))
X$
8
```

X$ = "8".

Character Conversions

If you look at the table in appendix C, you'll see that a decimal number corresponds to every character, symbol, and key. The numbers range from 0 through 255. There are three functions, `CHR$`, `NUM`, and `UPC$`, that enable you to convert a number to its corresponding character, convert a character to its corresponding decimal number character code, and convert small letters to capital letters.

Numbers to Characters

The `CHR$` *(character)* function converts a numeric value in the range –32768 through 32767 into a string character. Any number outside the range 0 through 255 is converted MOD256 to that range. Any number outside the range –32768 through 32767 is "ヒ" (the same as `CHR$(255)`).

`CHR$(`numeric expression`)`

Examples:

```
CHR$(35)
#
CHR$(126)
Σ
CHR$(16)
Θ
CHR$(8)
∆
CHR$(136)
∆̲
```

One of the most used numbers is 34 (this is the decimal number for a quotation mark). Often you may want to use the quotation mark in a `PRINT` or `DISP` statement. Since the beginning and end of a literal message is defined by a quotation mark, you cannot use the mark itself. Instead, use `CHR$(34)`:

```
DISP "The answer is, ";CHR$(34);
"YES";CHR$(34);"; you lose!"
The answer is, "YES"; you lose!
```

Characters to Numbers

The `NUM` (*numeric*) function converts an individual string character to its corresponding decimal value.

`NUM(`string expression`)`

Thus you can find the decimal number code of the corresponding character without having to look it up in the table in appendix C.

If more than one character is included in the string expression, the `NUM` function finds the decimal equivalent of the first character.

Examples:

```
NUM("¿")
 1
NUM("#")
 35
NUM("Θ")
 16
NUM("$^&*")
 36
```

To display ¿, type A while holding down the CTRL key (A^c).

To display Θ, type P while holding down the CTRL key (P^c).

Converts only first character of string.

Lowercase to Uppercase Conversion

The `UPC$` *(uppercase)* function enables you to convert a string with lowercase letters to a string composed of all uppercase letters.

`UPC$(`string expression`)`

Examples:

```
M$="yes"

N$=UPC$(M$)
N$
YES
UPC$("SOMEupsomeDOWN")
SOMEUPSOMEDOWN
```

Assigns M$ the string shown in lowercase letters.

Assigns N$ that string in uppercase letters.

Recalls N$.

The string need not be composed of all lowercase letters to be converted to all uppercase letters.

As you may have noticed from the table of characters in appendix C, lowercase letters have different decimal values than uppercase letters. The uppercase function allows strings to be compared without regard to upper and lowercase. For example, part of a program might be:

```
.
.
.
30 INPUT A$
40 IF UPC$(A$[1,1])="Y" THEN 80
```

User may enter `Y`, `y`, `yes`, `YES`, etc., and the program will branch to statement 80.

Assigning Values to Variables in a Program

You can assign values to variables using a program statement or by an input from the keyboard. Thus far, we have discussed the `LET` *(assignment)* statement and the `INPUT` statement with regard to simple variables. This section covers assignments to the elements of arrays, initializing variables, and three more statements that are used for assigning values to variables: `READ`, `DATA`, and `RESTORE`. These statements are useful when you have a large amount of data that is reused in different places in the program.

Assigning Values to Array Elements

Elements of an array are assigned values in the same manner as simple variables: from the keyboard or within a program. But a particular element must be referenced by its subscripts. For instance, M(1,2) refers to an element in array M and may be assigned a value and used in calculations like a simple variable.

151
155
159
163
167
171
152
156
160
164
168
172
153
157
161
165
169
170

Example:

```
10 OPTION BASE 1
20 DIM M(3,4)
30 LET M(1,2)=10
40 A=M(1,2)/7
50 PRINT M(1,2),A
60 END
```

Dimensions a 3 by 4 array M.
Assigns element M(1,2) the value 10.
You can use this element in calculations.

If we had not dimensioned array M, it would have been implicitly dimensioned with upper bounds of 10 for each subscript.

The program below enables you to input values from the keyboard. The `FOR-NEXT` loop is the most efficient means of manipulating array variables.

Example:

```
10 OPTION BASE 1
20 DIM A(5)
30 FOR I=1 TO 5

40 INPUT A(I)

50 NEXT I
60 FOR I=1 TO 5
70 PRINT "A(";I;")=";A(I)
80 NEXT I
90 END
```

You must assign each array element its value, individually.
Assigns the elements of array A the values you input.

Then prints the array elements.

Run the program, now, with the numbers 33, 48, –16, 3, and 10.

```
PRINTALL
RUN
?
33
?
48
?
-16
?
3
?
10
A( 1 )= 33
A( 2 )= 48
A( 3 )=-16
A( 4 )= 3
A( 5 )= 10
```

You can see that assigning values to array elements with a `FOR-NEXT` loop is indeed faster and easier than using an assignment statement for each element, especially with large arrays.

Let's see how this is done with two-dimensional arrays:

Example:

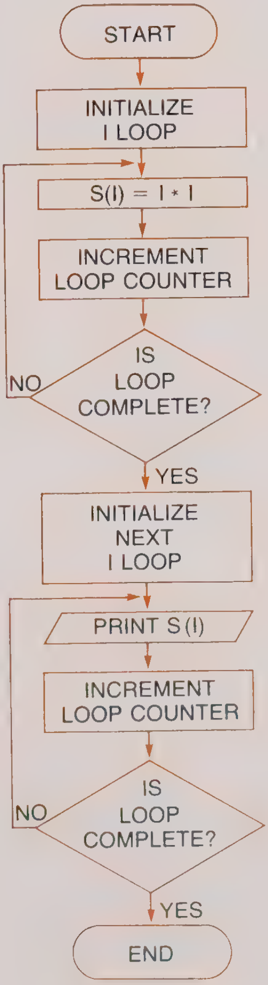
```
10 OPTION BASE 1
20 DIM K(3,5)
30 FOR I=1 TO 3
40 FOR J=1 TO 5
50 DISP "ROW";I;"COLUMN";J
60 INPUT K(I,J)
70 NEXT J
80 NEXT I
90 END
```

Lower bound of array is 1.
Dimension 3 by 5 array K.
Nested `FOR-NEXT` loops.
First input all elements of row 1, then all elements of row 2, etc.

There are many ways to assign array elements values within the program. The following program uses the loop counter to produce a list of squares of consecutive integers from 1 to 15.

```
10 OPTION BASE 1
20 DIM S(15)
30 FOR I=1 TO 15
40 S(I)=I*I
50 NEXT I
60 FOR I=1 TO 15
70 PRINT "S(";I;") =";S(I)
80 NEXT I
90 END
```

```
RUN
S( 1 ) = 1
S( 2 ) = 4
S( 3 ) = 9
S( 4 ) = 16
S( 5 ) = 25
S( 6 ) = 36
S( 7 ) = 49
S( 8 ) = 64
S( 9 ) = 81
S( 10 ) = 100
S( 11 ) = 121
S( 12 ) = 144
S( 13 ) = 169
S( 14 ) = 196
S( 15 ) = 225
```


START
INITIALIZE I LOOP
S(I) = I * I
INCREMENT LOOP COUNTER
IS LOOP COMPLETE?
NO
YES
INITIALIZE NEXT I LOOP
PRINT S(I)
INCREMENT LOOP COUNTER
IS LOOP COMPLETE?
NO
YES
END

Initializing Variables

It's good programming practice to initialize (set) variables to their starting values in a program before you use them. All numeric variables are initialized to undefined values by `RUN` or `INIT`. Thus, if you access the variable before it is defined, an error will occur.

As long as you assign the variable a value before it is accessed, you will not err. For instance, the following programs on the right cause warning messages (with `DEFAULT ON`) or error messages (with `DEFAULT OFF`) to occur. With `DEFAULT ON`, the default value of 0 is assigned to uninitialized variables, and the specified computations or programming operations will be performed. But a warning message is displayed to alert you to the error.

Correct

```
10 T=0
20 INPUT W
30 T=T+W
40 DISP T;W
50 END
RUN
?
5
 5   5
```

Incorrect

```
10 INPUT W
20 T=T+W
30 DISP T;W
40 END
```

Error here; using T when it has not yet been assigned a value.

```
RUN
?
5
Warning 7 on line 20 : NULL DATA
 5   5
```

Correct

```
10 DIM A(6,4)
20 A(3,3)= 3
30 DISP A(3,3)+3*2
40 END

RUN
 9
```

Incorrect

```
10 DIM A(6,4)
20 DISP A(3,3)+3*2
30 END
```

Error occurred here; using a variable that has not yet been assigned a value.

```
RUN
Warning 7 on line 20 : NULL DATA
A
 6
```

If you don't plan to assign values to all array elements in a program but want to be able to access any of them, you can easily initialize them using `FOR-NEXT` loops. For example, suppose we want to initialize all elements of array A to 0:

```
10 DIM A(6,4)
20 FOR I=0 TO 6
30 FOR J=0 TO 4
40 A(I,J)=0
50 NEXT J
60 NEXT I
.
.
```

The READ and DATA Statements

Many programs require you to enter large numbers of data items into the computer. You can accomplish this with the `INPUT` or `LET` statements, though it may be cumbersome to do so. If you had used an `INPUT` statement and decided to run the program with the same values at a later date, you would have to reenter all of the data once again. BASIC programming language provides a more convenient means of assigning values to variables in these instances—by using the `READ` and `DATA` statements.

The `READ` and `DATA` statements work together to assign values to variables within a program.

`READ` variable name$_1$ [`,` variable name$_2$ ...]
`DATA` constant or string [`,` constant or string ...]

The `READ` statement specifies the variables whose values are to be assigned from within the program. The variables in a `READ` statement may be simple variables, subscripted variables, or string variables, and they must be separated by commas.

The `DATA` statement contains a list of the numbers or character strings that will be assigned to the variables in the `READ` statement. The numbers or strings must be separated by commas. Each `DATA` item must correspond to the appropriate variable of the same type in a `READ` statement.

```
10 READ N$,A(1),C
20 IF C>3 THEN PRINT N$;A(1);C
30 DATA "NAME", 43,6
40 END
RUN
NAME 43  6
```

These statements cause "NAME" to be assigned to N$, 43 to A(1), and 6 to C.

In `DATA` statements, literal text may be quoted *or* unquoted.

The `DATA` statement is declaratory and is simply ignored in a program if there is no corresponding `READ` statement. Therefore, a `DATA` statement need not correspond exactly with the `READ` statement. Your `DATA` statements can contain more items than accessed by the `READ` statement, and they can be positioned anywhere in a program except after `THEN` or `ELSE` in an `IF... THEN` statement. The important point is that the order of `DATA` statements within a program determines the order of their use.

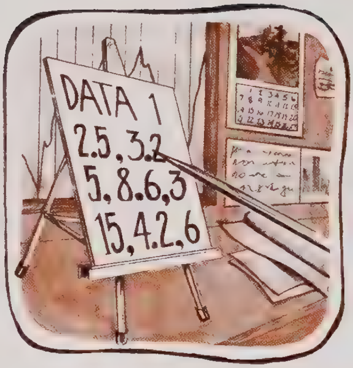
For example, load the following program and run it:

```
10 DATA 24,8.3,17,19,3.2,58
20 FOR I=1 TO 4
30 READ X
40 PRINT X; "SQUARED=";X^2
50 NEXT I
60 END
```

Extra data items are ignored.

```
RUN
24 SQUARED= 576
8.3 SQUARED= 68.89
17 SQUARED= 289
19 SQUARED= 361
```

The system uses an internal mechanism, called a "pointer," to locate the data element that is to be read. The left-most element of the lowest-numbered DATA statement is read first. After this element is read, the data pointer repositions itself one element to the right and continues to do so each time another data item is read.



After reading the last element in a DATA statement, the data pointer locates the next higher-numbered DATA statement (if any) and repositions itself at the first element in that statement. But if there are no higher-numbered DATA statements, the data pointer remains at the end of the last DATA statement; any effort to read additional data will cause a NO DATA message to be displayed.

For example:

```
AUTO 10
10 READ N
20 FOR P=1 TO N
30 READ D,D1
40 DISP D^2-D1
50 NEXT P
60 DATA 4
70 DATA 9,1,8,4,7,9
80 END
```

Assigns 4 to N.

First, assigns 9 to D and 1 to D1; then, 8 to D and 4 to D1; finally 7 to D and 9 to D1.

```
RUN
80
60
40
Error 34 on line 30 : NO DATA
```

Since N=4, program tries to read more values for D and D1, but finds no more data available.

The DATA statement in the last program can be entered in a variety of ways. For example, the following representations are equivalent:

Note: We do not change the order of the items themselves.

```
60 DATA 4,9,1,8,4,7,9
```

or

```
60 DATA 4,9
62 DATA 1,8
64 DATA 4,7
66 DATA 9
```

or

```
65 DATA 4,9,1
68 DATA 8,4,7,9
```

Even though the data items can be entered in one or several DATA statements, as shown above, the order in which they appear must correspond exactly with the order in which you access them.

The READ and DATA statements are often used to assign values to array elements.

Example:

```
10 FOR I=1 TO 5
20 FOR J=1 TO 5
30 READ A(I,J)
40 NEXT J
50 NEXT I
60 DATA 1,2,3,4,5,2,4,6,8,10,3,6
,9,12,15,4,8,12,16,20,5,10,15,20
,25
70 FOR I=1 TO 5
80 FOR J=1 TO 5
90 PRINT A(I,J);
100 NEXT J
110 PRINT
120 NEXT I
130 END
```

```
RUN
1  2  3  4  5
2  4  6  8  10
3  6  9  12  15
4  8  12  16  20
5  10  15  20  25
```

Notice that you must READ each array element one at a time.

The semicolon causes printed items to be retained in the print buffer.
The extra print statement forces a print after each row.

You can see a number of things about READ and DATA from the examples above. To recap:

- It doesn't matter where the DATA statement is, in relation to the READ statement, as long as the data items correspond to the variables in the READ statement in order and in type.
- More than one READ statement can access a DATA statement. As each READ is executed, the DATA pointer moves to the next data item. There must be at least as many items in the set of DATA statements as there are variables in the READ statements. Extra data items are ignored.
- The items in a data list must be either numbers or strings; variables and formulas are not allowed. A data item accessed by a string variable in a READ statement may be quoted or unquoted in the DATA statement.
- Variable assignments made with READ and DATA statements are part of a program, contrasted with variable assignments made with the INPUT statement. Thus, the data is stored with the program and will remain with the program until the DATA statement, itself, is changed.

Rereading Data: The RESTORE Statement

Up to this point we have been able to access DATA items only once in a program. Once the data pointer moves past the last data item in the last DATA statement, an additional READ statement causes "Error : NO DATA". Of course, if the data items have been assigned to variables, you can use the same values again by using the variable name.

But certain programs may require some, if not all, of the data to be read more than once. BASIC provides the RESTORE statement just for this purpose:

RESTORE [statement number]

The `RESTORE` statement resets the data pointer to the first item of the specified statement (or the first item of the lowest-numbered `DATA` statement in the program if no statement is specified) each time it occurs within a program.

For example:

```
10 READ A,B,C$
20 IF C$#"N" THEN 50
30 RESTORE
40 GOTO 10
50 PRINT A,B
60 GOTO 10
70 DATA 5,10,"Y",15,20,"Y",3,9,"
   N"
80 END
```

```
RUN
5                10
15               20
5                10
15               20
5                10
15               20
5                10
15               20
```

PAUSE

As you can see, the data pointer is continually reset to 5 in the `DATA` statement each time "N" is read and `RESTORE` is executed.

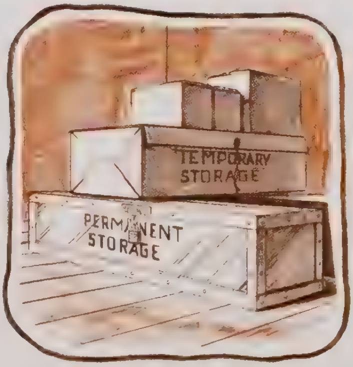
System Memory and Variable Storage

Memory

The HP-85 uses two types of memory: **read/write memory** (or random access memory—RAM) and **read only memory** (ROM). Read/write memory is used to store programs and data. When you store a program or data, you "write" into memory. When you access a line of your program or data element, you "read" from memory, thus the term read/write. Read/write memory is temporary; it can be changed or erased.

Programs and data in read/write memory can be saved for future use by recording the information on a tape cartridge or other storage medium.

Read only memory differs in that it is permanent. When the machine is turned off, the contents of the read/write memory are lost, whereas the read only memory is unaffected. ROM modules can be plugged into the slots in the back of the machine, making it possible to expand the language and capabilities. A small amount of read/write memory is used by some plug-in ROMs. This area is called "working storage."



Storing Variables

Byte is computer language for a "memory location" composed of eight *bits* (*b*inary dig*its*). It is the basic unit of information to or from memory. A kilobyte is a unit of 1,024 bytes and is abbreviated as "K".

The HP-85 has 16K bytes (or 16,384 bytes) of read/write memory; 14,576 are available for your use. The memory can be expanded to 32K bytes of which 30,704 are available for your use with the memory module; refer to appendix A.

Use the following tables to determine the number of bytes that variables need in order to be stored in system memory. (Do not confuse storing in system memory with storing on a tape cartridge. Tape cartridge storage will be discussed in section 11.)

Simple Variables	Bytes of Memory
Full precision	10 bytes
Short precision	6 bytes
Integer	5 bytes
String	8 bytes + 1 byte per character

Array Variables	Bytes of Memory
Full precision	8 bytes + 8 bytes per element
Short precision	8 bytes + 4 bytes per element
Integer	8 bytes + 3 bytes per element

You have already noticed that at the end of every program listing, the HP-85 displays the number of bytes (or memory locations) remaining in system memory. Press INIT or execute the `INIT` (*initialize*) command before `LIST` or `PLIST` so that the memory displayed will include the memory required for allocated variables.

If you do not wish to `LIST` the entire program to recall the memory, type `LIST` and then a statement number larger than any in the current program. For instance, you could execute `LIST 9999` to display the number of bytes left.

You need not have a program in memory to execute `LIST`. If there is no program, the system merely outputs the number of bytes available.

Conserving Memory

Large programs that involve large amounts of data sometimes need more memory than is available for use. You can conserve memory by:

1. Limiting the use of `REM` statements and comments in a program. This limits program readability and documentation, but it does conserve memory.
2. Using `SHORT` and `INTEGER` precision array variables, whenever possible or convenient, rather than full precision. This is a very good way to conserve memory in a program that has a lot of data and is most useful when dealing with large arrays.
3. A third way to conserve memory is to break a program down into several sections and `STORE` each section into a different file. Then each section of the program can be brought into memory, one at a time, using the `CHAIN` statement. (Refer to section 11.)

4. Combine statements using "`@`". This reduces program readability, but it does conserve memory by three bytes per line. For example:

```
10 BEEP @ BEEP
```

is seven bytes of information while

```
10 BEEP
20 BEEP
```

is 10 bytes of information.

Problems

8.1 Here is your chance to invent some new words. Write a program that accepts a base string and a first-letter string, and then prints the "words" formed by combining each of the first letters with the base string, but omitting those that would begin with a double letter.

8.2 One light-year is the distance light travels in one year—approximately 9 trillion kilometers. The distances (in light-years) of the 27 stars within 15 light-years of our solar system are listed below. Write a program that will group these distances into intervals of 1 light-year (0-1 through 14-15) and determine the number of stars in each interval. After printing these results, the program should request an interval number, 1 through 15, and print the distances for the stars in that interval. Use an `INTEGER` array for accumulating the interval distributions. Use a simple `SHORT` variable for the actual distances and `READ` them one at a time. A `RESTORE` statement is necessary for the second part of the program.

STAR DISTANCES (light-years)			
4.3	10.3	11.5	12.8
5.9	10.7	11.6	13.1
7.6	10.8	11.7	13.1
8.1	10.8	11.9	13.9
8.6	11.2	12.2	14.2
8.9	11.2	12.5	14.5
9.4	11.4	12.7	

8.3 The world record, set in 1970, for the 30-kilometer run is 1:31:30.4 (1 hour, 31 minutes, 30.4 seconds) and is held by Jim Adler of Britain. In 1974, Bernd Kannenberg of West Germany set a world record of 2:12:58.0 for the 30-kilometer walk. Write a program that accepts an individual's time for a 30-kilometer course and calculates the average speed according to

$$\text{Speed (m/s)} = \frac{30{,}000\ \text{(m)}}{\text{Time (s)}}$$

The time is to be specified in *hours:minutes:seconds* format (including colons). Use the `POS` function to locate the colons, and the `VAL` function to extract the numerical values from the string. Also, use the program to calculate the speed of Sergei Saveliev of the USSR, who set a world record of 1:30:29.38 for the 30-kilometer Nordic ski event in 1976, and for Clem Turvy on his motorcycle, covering 30 kilometers in 26:44 (26 minutes, 44 seconds).

8.4 Although a string variable may not be declared to be an array, it is possible to use substrings of a string variable to achieve the effect of a "string array." For example, if the words representing the numbers 0 through 9 are strung together with proper spacing, any one word is readily accessible by determining the first and last substring specifiers corresponding to the word (similar to the subscript of an array element). Using this concept and concatenation, write a program that counts from 0 to 99 in this way:

```
ZERO
ONE
.
.
NINE
ONE   ZERO
ONE   ONE
.
.
NINE  NINE
```

8.5 Farmer Flem Snopes wants to install irrigation sprinklers in his three strawberry patches. The table below gives coverage diameters for a particular sprinkler design at various water pressures and nozzle options. Write a program based on this table that asks for the width of the irrigated strip (which determines the minimum coverage diameter) and the available water pressure at that location, and then specifies the appropriate nozzle option. Use the program to find the nozzle options for Snopes' east strawberry patch (150 feet wide, 75 psi pressure), his southeast patch (140 feet wide, 75 psi pressure), and his far-north patch (140 feet wide, 60 psi pressure).

Coverage Diameter (feet)

Nozzle Option	A	B	C	D
Water Pressure (psi)				
60	124	133	138	142
65	126	136	141	146
70	129	139	144	149
75	132	142	147	152
80	134	145	150	155

Section 9

More Branching

There's much more to branching operations on the HP-85 than `IF` ... `THEN` and `GOTO`. The system enables you to define your own functions and use them in programs, just as you use the built-in functions. For longer program segments or routines that are often repeated within a program, the BASIC language provides subroutines that can be accessed any number of times within a program. In addition, the system contains three timers that can interrupt a program in the time intervals of your choice. Last, but not least, we'll discuss the special function keys—how to define them so that when pressed, they immediately cause special branching in a program.

Defining a Function

If a numeric or string operation has to be evaluated several times, it is convenient to define it as a function. With the `DEF FN` *(define function)* statement, you can define your own functions within a program and reference them in exactly the same manner that you reference the system's built-in functions. A function must be defined in the same program that references the function. The definition can appear anywhere in the program, before or after the function is referenced.

`DEF FN`numeric variable name [`(`parameter `)`] [`=` numeric expression]
`DEF FN`string variable name [`(`parameter `)`] [`=` string expression]

Once a function is defined, it can be used by referring to the function name. A numeric function name must consist of the letter `FN` followed by a numeric variable name. A string function name is a numeric function name followed by a dollar sign, `$`. If the function requires an argument, then it must appear immediately after the function name, enclosed within parentheses. The parameter may be any simple numeric or string variable name. Array names are not allowed. The length of a string argument passed between a function and the main program defaults to 18 characters. But you *can* allocate a larger string in the function definition. Refer to page 151.

Single-Line Functions

The simplest form of a function definition is the single-line function. The function is defined in one `DEF FN` statement with an equals sign separating the function name from the expression assigned to the function.

For example, the following program defines `FNX2` as the X^2 function and then uses the function to evaluate 8^2.

```
10 REM *X SQUARED
20 DEF FNX2(N)=N*N
30 DISP FNX2(8)
40 END

RUN
 64
```

Defines function FNX2.
Displays FNX2(8).

The parameter, `N`, in statement 20 is a dummy variable used only in the definition. It is replaced by the actual variable or expression when used to evaluate the function. In this case, `N` is replaced by 8.

All user-defined functions may have, at most, one argument. The function is evaluated using that argument to return, at most, one value at a time.

But a function need not have an argument. (Recall the `PI`, `EPS`, and `INF` built-in functions.)

Examples:

```
10 DEF FNE$="10-SECONDS"
20 DISP FNE$
30 END
RUN
10-SECONDS
```

```
10 REM *Planck's constant
20 DEF FNH=6.625E-27
30 PRINT FNH;FNH^2
40 END
RUN
6.625E-27   4.3890625E-53
```

A function definition cannot be recursive; in other words, you may not use the function that you are defining in the expression that defines the function or in any user-defined function referenced by that expression. But you may use any other user-defined function that has been fully defined elsewhere in the program, and of course, you can use any of the built-in functions in the definition.

Example: Write a program that defines function `FNR` to round any given number to the hundredths place. Then use `FNR` to display the square roots of 1, 1.5, 2, 2.5, … , 10.

```
10 REM *ROUND TO 2 DECIMAL PLAC
   ES
20 FOR I=1 TO 10 STEP .5
30 DISP I,FNR(SQR(I))
40 NEXT I
50 DEF FNR(D2) = INT(D2*100+.5)
   /100
60 END
```

Notice the use of non-integer steps.

Defines rounding function.

```
RUN
 1                    1
 1.5                  1.22
 2                    1.41
 2.5                  1.58
 3                    1.73
 3.5                  1.87
 4                    2
 4.5                  2.12
 5                    2.24
 5.5                  2.35
 6                    2.45
 6.5                  2.55
 7                    2.65
 7.5                  2.74
 8                    2.83
 8.5                  2.92
 9                    3
 9.5                  3.08
 10                   3.16
```

Displays square roots of number in left column, rounded to hundredths place.

A function definition is a declaratory statement and may be placed anywhere in the program. It merely defines the function, and is ignored by the program unless it is referenced elsewhere by the function name.

See problems 9.1 through 9.3 at the end of this section for more examples of single-line functions.

Multiple-Line Functions

Often, a single line is not enough to define a function, especially if the function contains lengthy computations or branching operations. Multiple-line functions work much like single-line functions in that the function can contain at most one argument and returns one value. Again, the function definition may be placed anywhere within the program since, as a block of statements, it is non-executable unless it is referenced by the function name. But more sophisticated functions can be defined since a function can be described in a series of statements.

There are three basic parts to the multiple-line function definition:

1. The first statement is the `DEF FN` statement. It is the only `DEF FN` statement that may occur within the function definition.
2. The last statement is the `FN END` *(function end)* statement.
3. At least one of the statements in the function definition should assign the function name a value.

Unlike single-line functions, the function definition is not included in the `DEF FN` statement. Only the function name and argument (if any) must be declared.

The `FN END` *(function end)* statement defines the end of a multiple-line function. Its syntax is simply:

```
FN END
```

Any number of statements can be included between the `DEF FN` and `FN END` statements. But one of these statements should assign the final value of the function to the function name.

For example, this program defines a function that converts an integer with a decimal base to its octal equivalent.

● `10 DEF FNO(D)`	Defines beginning of multiple line function.
`20 D=IP(D)`	Throw away the fractional part of the number to avoid an error.
`30 N8=0` `40 I=1`	Initializes variables N8 and I.
`50 Q=IP(D/8)` `60 N8=N8+(D-Q*8)*I`	Converts decimal value to octal equivalent.

`70 D=Q`
`80 I=I*10`
`90 IF D#0 THEN 50`
	Works for both positive and negative integers.

● `100 FNO=N8`	Assigns function name a value.
● `110 FN END`	Function end.

The dots by statements 10, 100, and 110 indicate the essential parts of a multiple-line function.

Again, the program segment above only *defines* the function. In order to *evaluate* the function, you must reference it in another part of the same program, replacing the parameter D with the desired expression.

For instance, add the following statements to the program segment above.

```
   1 PRINT "DECIMAL","OCTAL"
   2 FOR J=128 TO 256
   3 PRINT J,FNO(J)
   4 NEXT J
 120 END
```

RUN

```
DECIMAL                 OCTAL
 128                     200
 129                     201
 130                     202
 131                     203
 132                     204
 133                     205
 134                     206
 135                     207
 136                     210
 137                     211
 138                     212
 139                     213
 140                     214
 141                     215
 142                     216
 143                     217
 144                     220
```

STEP
PAUSE

Notice that we used the variable J as our loop counter in program statements 2 through 4. What if we had used the variable I in both our main program and in the function definition?

```
   1 PRINT "DECIMAL","OCTAL"
   2 FOR I=128 TO 256
   3 PRINT I,FNO(I)
   4 NEXT I
  10 DEF FNO(D)
  20 D=IP(D)
  30 N8=0
  40 I=1
  50 Q=IP(D/8)
  60 N8=N8+(D-Q*8)*I
  70 D=Q
  80 I=I*10
  90 IF D#0 THEN 50
 100 FNO=N8
 110 FN END
 120 END
```

This program would only generate the first value of FNO(I) because the value of I is changed in the function definition.

Note that variable D in the main program would not be similarly affected. For instance, if all of the I variables were changed to D's in statements 1 through 4, the program would work.

The program would not compute all of the values assigned by the loop counter.

```
DECIMAL                 OCTAL
 128                     200
```

The point of our discussion, here, is that if the value of a variable is changed in a function definition, then it is also changed in the main part of the program.

Let's look at two examples of multiple-line functions using string variables.

Example: Write a program that formats a number with a comma in place of the decimal point. If the number is an integer, supply two zeros to the right of the comma. Consider only numbers with absolute values that are greater than 1 × 10^{-11} and less than 1 × 10^{11}.

```
   10 REM *REPLACE POINT WITH COMM
      A
 • 20 DEF FNE$(N)
   30 IF FP(N)=0 THEN F$="000" ELS
      E F$=VAL$(ABS(FP(N)))
   40 I$=VAL$(IP(N))
 • 50 FNE$=I$&","&F$[2]
 • 60 IF ABS(N)<.00000000001 OR AB
      S(N)>100000000000 THEN FNE$=
      "OUT OF RANGE"
 • 70 FN END
   80 INPUT D
   90 DISP D,FNE$(D)
  100 GOTO 80
  110 END
```

Function definition.
Checks for out-of-range numbers.

RUN

```
?
1234.1234
 1234.1234                 1234,1234
?
56.85
 56.85                     56,85
?
14
 14                        14,00
?
689234.156
 689234.156                689234,156
```

PAUSE

Example: Now write a program that formats a number with commas every three digits to the left of the decimal point, using a multiple-line string function to insert the commas. Consider only numbers with absolute values less than or equal to 1 × 10^{11} and greater than 1 × 10^{-11}.

```
 10 REM *INSERT COMMAS
 20 DEF FNC$(N)
 30 B=0 @ N$="" @ M=1000000000 @
     N1=N
 40 IF ABS(N)<=100000000000 AND
    ABS(N)>.00000000001 THEN 70
 50 FNC$="OUT OF RANGE"
 60 GOTO 160
 70 FOR I=1 TO 3
 80 IF N1<M THEN 120
 90 B=IP(N1/M)
100 N1=FP(N1/M)*M
110 N$=N$&VAL$(B)&","
120 M=M/1000
130 NEXT I
140 N$=N$&VAL$(N1)
150 FNC$=N$
160 FN END
170 INPUT X
180 DISP X,FNC$(X)
190 GOTO 170
200 END
```

Function definition.

```
RUN
?
1234
 1234                 1,234
?
1234567
 1234567              1,234,567
?
12345678912
 12345678912
12,345,678,912
?
1234.567
 1234.567             1,234.567
?
123456789.12
 123456789.12
123,456,789.12
?
```

PAUSE

Multiple-line functions are not recursive. For example, the following attempt to define a factorial function would generate an error message.

```
10 DEF FNF(X)
20 IF X=0 THEN FNF=1 ELSE FNF=X
   *FNF(X-1)
30 FN END
40 INPUT T
50 PRINT T;FNF(T)
60 GOTO 40
```

The error occurs in attempting to use the function name in the function definition.

```
RUN
?
3
Error 42 on line 20 : RECURSIVE
FN CALL
```

As we mentioned earlier, the length of a string argument passed between a function and the main program defaults to 18 characters. You can specify a larger string in the function definition by enclosing the length within brackets following the string argument. For instance:

`DEF FNS$(A$[75])`	Allocates a string argument of 75 characters.

You cannot use the `DIM` statement to dimension the string argument since it is considered a "dummy" variable. Therefore, you must allocate space for the argument within the `DEF FN` statement itself. When you do this, the system considers the entire `DEF FN` statement, including the allocated variable, as part of the program line. Thus, the maximum length of the string argument in a multiple-line function is approximately 240 characters and the maximum length of the string argument in a single line function is dependent on the complexity of the expression that defines the function. If the argument is too large, the system will display `Error 85 : EXPR TOO BIG`. If this happens, decrease the length of the string argument until the system accepts the statement.

Note that although a string function may accept a string argument larger than the default length, the resulting string passed from the function to the main program can be no longer than 18 characters.

Refer to problems 9.4 through 9.6 at the end of this section for more examples of multiple-line functions.

Subroutines

Often, the same sequence of statements is executed more than once within a program. By using a subroutine you can key in the group of statements only once and then access the statements from different places within the program. If you group all of the often-used routines at the end of your program, you can make the program easier to follow and understand.

Subroutines are similar to functions in that they can be referenced from other parts of the program. But a subroutine is not given a name; it is referenced by a `GOSUB` statement and the beginning statement number of the routine.

`GOSUB` statement number

The `GOSUB` statement transfers program control to the subroutine you wish to execute. The statement number must be that of the first statement of the subroutine.

A subroutine can begin with any statement except `NEXT`. For example, the subroutine might begin with `REM`, `LET`, `IF... THEN`, `FOR`, etc. The last statement of a subroutine must be a `RETURN` statement.

`RETURN`

There may be more than one `RETURN` statement within a subroutine. As soon as a `RETURN` statement is encountered, program control is transferred to the statement following the particular `GOSUB` that referenced the routine.

Arguments or parameters are not used to pass values from the subroutine to the main program. As with functions, all variables used in subroutines are global variables, in other words, all program variables are accessible in both functions and subroutines. If the value of the variable is changed within a subroutine, it is also changed in the main program. Unlike user-defined functions, subroutines cannot appear in a numeric or string expression. And, character strings "passed" between a subroutine and the main program can be as long as the length dimensioned in a `DIM` statement.

For example:

```
  10 DISP "ENTER NUMBER"
  20 INPUT N
  30 IF N<0 THEN 10
• 40 GOSUB 100
 .
 .
 .
  90 GOTO 160
 100 REM *SUM FROM 1 TO N
 110 S=(N*(N+1))/2
 120 PRINT "SUM=";S
 .
 .
 .
 150 RETURN
 160 N=N*2
•170 GOSUB 100
 .
 .
 .
 300 END
```

Subroutine.

When the program executes statement 40, program control is immediately transferred to statement 100. When a `RETURN` statement is encountered, control is transferred to the line following 40. Statement 170 also transfers control to statement 100. In this case, `RETURN` transfers program control to the line following 170.

Subroutines may be nested, that is, a second subroutine can be entered before the `RETURN` statement of the first is executed.

For example:

```
  10 DISP "ENTER NUMBER"
  20 INPUT N
  30 IF N<0 THEN 10
• 40 GOSUB 1000
 .
 .
 .
  90 STOP
 1000 REM *SUM FROM 1 TO N
 1010 S=(N*(N+1))/2
 1020 PRINT "SUM =";S
 1030 DISP "SUM OF SQUARES (Y/N)"
 1040 INPUT A$
•1050 IF A$ = "Y" THEN GOSUB 2000
 .
 .
 .
 1200 RETURN
 2000 REM *SUM SQUARES OF INTEGER
      S FROM 1 TO N
 2010 S2=(N*(N+1)*(2*N+1))/6
 2020 PRINT "SUM OF SQUARES=";S2
 .
 .
 .
 2090 RETURN
```

Nested subroutine.

The subroutine at line 2000 is nested within the one at line 1000. The RETURN statement on line 2090 returns to the line following 1050 in the first subroutine. The RETURN statement at 1200 returns to the line following statement 40.

Subroutines can be nested as deeply as available memory allows (up to 255 levels of nesting). When a RETURN is executed, control returns to the subroutine that was entered most recently.

See problem 9.7 to write a complete program that uses subroutines.

The Computed GOSUB Statement

The ON... GOSUB *(computed GOSUB)* statement enables you to access any of one or more subroutines based on the value of a numeric expression. It operates exactly like an ON... GOTO statement except that instead of transferring program control to one statement, ON... GOSUB transfers control to the first statement of a subroutine. The RETURN statement of the subroutine returns program execution to the statement following the ON... GOSUB statement that referenced it. The ON... GOSUB statement is programmable only; it can't be executed from the keyboard.

ON numeric expression GOSUB statement number list

The numeric expression is evaluated and *rounded* to an integer. A value of 1 causes the subroutine at the first statement number in the list to be accessed; a value of 2 causes the subroutine at the second statement number in the list to be accessed, and so on.

All RETURN statements in the subroutines accessed transfer program control back to the end of the statement number list of the ON... GOSUB statement.

For example:

```
 10 FOR X=1 TO 3
 20 ON X GOSUB 200,300,400
 30 NEXT X
.
.
.
100 STOP
200 PRINT X,SIN (X)
290 RETURN
300 PRINT X,X^2,COS(X
390 RETURN
400 PRINT X,X^3;TAN(X)
490 RETURN
```

This statement means:
If X=1, then GOSUB 200.
If X=2, then GOSUB 300.
If X=3, then GOSUB 400.
Program control reaches statement 40 when the FOR-NEXT loop is completed. RETURN in each subroutine transfers control to statement 30.

Subroutine 200.

Subroutine 300.

Subroutine 400.

If the value of the numeric expression is less than one or greater than the number of statement numbers in the list, an error occurs.

Problem 9.8 provides another example of the use of the ON... GOSUB statement.

Branching Using Special Function Keys

You have seen some of the many uses of the special function keys from running the programs in the Standard Pac. The eight special function keys, k1 through k4 (unshifted), and k5 through k8 (shifted), can be used to interrupt a running program and cause branching.

This interrupt capability is declared with an `ON KEY#` statement. The `ON KEY#` statement specifies the branching operation that will occur when the related key is pressed.

`ON KEY#` key number [, key label] `GOTO` statement number
`ON KEY#` key number [, key label] `GOSUB` statement number

The key number must be an integer from 1 through 8. The "key label" is a string expression which is truncated to the first eight characters. When a user-defined key is pressed during a program run, and an `ON KEY#` statement has been declared for it, the specified branching occurs. With `ON KEY# . . . GOSUB`, *program control returns to the next executable statement.*

If a program is not running, pressing a user-defined key does nothing.

KEY LABEL

The `KEY LABEL` statement is used to recall key labels for the user-defined keys to the display. The statement is simply:

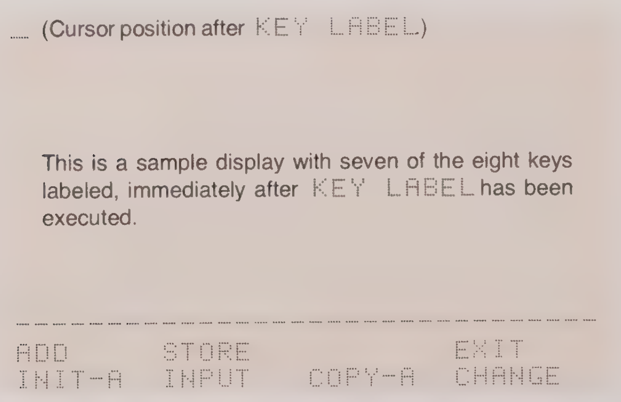
`KEY LABEL`

As you can see from the `ON KEY#` statement syntax, you can optionally specify a key label in the program definition of a key. Once defined and labeled in a program, the `KEY LABEL` statement causes the labels to appear on the lower three lines of the display.

All eight user-defined keys can have labels defined and displayed: each one appears in a unique location on the display, situated directly above the corresponding special function keys on the keyboard.

The KEY LABEL key recalls all current labels, at any time, and displays them on the bottom three lines of the display. It performs the same operation that the `KEY LABEL` statement does in a program.

Both the KEY LABEL and the `KEY LABEL` statement also move the cursor to the home position on the display. Thus a full 13 lines may be entered or displayed before the key labels are over-written.



_ (Cursor position after `KEY LABEL`)

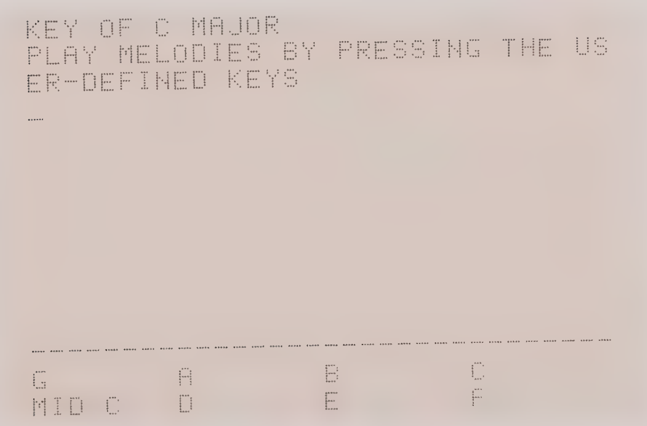
This is a sample display with seven of the eight keys labeled, immediately after `KEY LABEL` has been executed.

```
ADD      STORE             EXIT
INIT-A   INPUT    COPY-A   CHANGE
```

Perhaps the following short program can best illustrate the ease with which function keys can be defined and the rapidity with which they are executed when pressed while the program is running.

```
 10 ON KEY# 1,"MID C" GOSUB 100
 20 ON KEY# 2,"D" GOSUB 200
 30 ON KEY# 3,"E" GOSUB 300
 40 ON KEY# 4,"F" GOSUB 400
 50 ON KEY# 5,"G" GOSUB 500
 60 ON KEY# 6,"A" GOSUB 600
 70 ON KEY# 7,"B" GOSUB 700
 80 ON KEY# 8,"C" GOSUB 800
 90 CLEAR @ KEY LABEL
 96 DISP "KEY OF C MAJOR"
 97 DISP "PLAY MELODIES BY PRESS
    ING THE USER-DEFINED KEYS"
 98 GOTO 98
100 BEEP 201,100
110 RETURN
200 BEEP 178,100
210 RETURN
300 BEEP 157,100
310 RETURN
400 BEEP 147,100
410 RETURN
500 BEEP 130,100
510 RETURN
600 BEEP 114,100
610 RETURN
700 BEEP 101,100
710 RETURN
800 BEEP 94,100
810 RETURN
1000 END
```

As soon as you press (RUN), the display is cleared and the key labels are recalled:



```
KEY OF C MAJOR
PLAY MELODIES BY PRESSING THE US
ER-DEFINED KEYS
_











--------------------------------
G        A        B        C
MID C    D        E        F
```

Play a few tunes with the special function keys. The program quickly illustrates that each press of a special function key causes *one* execution of the GOTO/GOSUB as defined by the ON KEY # statement, and that one key interrupts another. When a defined key is pressed during a running program, the current program line is completed before the specified branching occurs.

Notice statement 98 in the Key of C program:

```
98 GOTO 98
```

Since `ON KEY#` statements are only active when a program is running, it is often necessary to have a place in the program that does nothing but idle, waiting for a keystroke. We cannot use a `STOP` or `END` statement to separate the key definitions from the subroutines; program execution would halt as soon as either statement was encountered. Thus, a `GOTO` statement that "goes to" itself keeps the `ON KEY#` declaratives active in a particular part of a program.

`ON KEY#` declaratives are temporarily deactivated while a program is waiting for a response to an `INPUT` statement. Pressing them on input will cause their related keycodes to appear on the input line. Key definitions are also deactivated after `PAUSE` is executed. They resume functioning with `RUN` or `CONT`. If another program is "chained" to the program with the `ON KEY#` statements, the key definitions will no longer be active. (Refer to the `CHAIN` command in section 11.)

Cancelling Key Assignments

The `ON KEY#` declarative holds for a key until another declarative for the same key, SCRATCH, or `OFF KEY#` is executed.

`OFF KEY#` key number

The `OFF KEY#` statement cancels the definition and branching operation of the specified key.

Problem 9.9 provides another example of `ON KEY#` statements. Refer to any of the Standard Pac programs for more examples.

The Timers

Along with the `SETTIME` statement and the time functions, the HP-85 provides three individual timers that may be set to interrupt a program at the specified time interval and cause the specified branching to occur. Interrupt intervals for the timers are declared with `ON TIMER#` statements. The `ON TIMER#` statement must be declared within a program.

`ON TIMER#` timer number `,` milliseconds `GOTO` statement number
`ON TIMER#` timer number `,` milliseconds `GOSUB` statement number

The timer number must be either 1, 2, or 3. The number of milliseconds must be a value less than $|99999999|$ and greater than $|.5|$. The sign of the milliseconds parameter is ignored. Zero and numbers outside the given range interrupt immediately and then wait 99999999 milliseconds before the next interrupt.

When the interrupt occurs, the specified branching occurs within a program.

For example, timer #1 interrupts the program every 15 minutes to go to statement 50 in the following program (as long as the program is running).

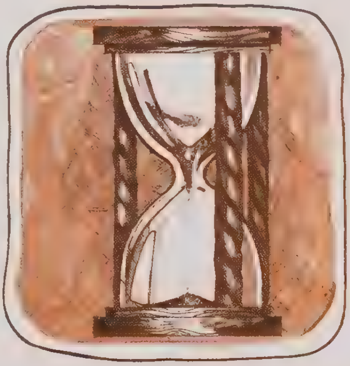
```
10 ON TIMER#1,900000 GOTO 50
:
:
50 BEEP 157,50 @ BEEP 201,50
60 BEEP 178,50 @ BEEP 272,75
70 WAIT 100
80 BEEP 272,50 @ BEEP 178,50
90 BEEP 157,50 @ BEEP 201,100
```

The timers continue to interrupt the system after a program is halted, but the interrupt does not cause the specified branching. The timers are deactivated when you edit the program, when SCRATCH or RESET is pressed, or when the `OFF TIMER#` statement is declared.

`OFF TIMER#` timer number

The `OFF TIMER#` statement deactivates the corresponding `ON TIMER#` statement. No further interrupts will occur from the specified timer until it is reactivated.

Example: Suppose you have written a lengthy program which is actually composed of five separate tests. Set up timers to wait for an input response from the user. If there is no response within 20 seconds, go to the next segment of the program.



```
 10 PRINT "SECTION 1.1"
 20 PRINT
 30 DISP "IF THIS TEST IS TO BE
    RUN, TYPE YES"
 40 ON TIMER# 1, 20000 GOTO 800
 50 INPUT Z9$
 60 OFF TIMER# 1
 70 IF Z9$#"YES" THEN 810
 80 PRINT
 90 PRINT "BEGIN TEST NOW"
100 PRINT
 .
 .
 .
```

If the user types "YES" within 20 seconds, the test will be executed. If there is no response to line 50 within 20 seconds, the program branches to statement 800.

```
800 OFF TIMER# 1
810 PRINT "SECTION 1.2"
820 PRINT
830 DISP "IF THIS TEST IS TO BE
    RUN, TYPE YES"
840 ON TIMER# 1, 20000 GOTO 1600
850 INPUT Y8$
860 OFF TIMER# 1
870 IF Y8$#"YES" THEN 1610
880 PRINT
890 PRINT "BEGIN TEST NOW"
 .
 .
 .
```

Reset timer for second test.

And so on.

The fact that timer's continue to interrupt even after the program is halted is important. Errors may occur if the timers are interrupting so fast that the system (program) cannot get anything done. Try this:

```
 10 ON TIMER# 1, 1 GOSUB 100
 20 ON TIMER# 2, 1 GOSUB 100
 30 ON TIMER# 3, 1 GOSUB 100
 40 STOP
100 DISP "SUB"
110 RETURN
120 END
```

First press RUN, then press LIST. When you press RUN the first timer tries to go to statement 100 but gets interrupted by the second timer and the second timer gets interrupted by the third, etc. Thus statement 100 may never be executed or the system will give you an error message. You'll find that the system will list the program very slowly since it is being interrupted continually. Execute SCRATCH, RESET, or the `OFF TIMER#` statement to halt the timers.

Refer to the Standard Pac for more examples using the timers, especially the Timer Program.

Problems

9.1.a. Define a single-line function that rounds a number at the decimal point. Evaluate the function from −5 to 5 in intervals of 0.3.

b. Define another single-line function that rounds a given number to the thousandths decimal place. Evaluate this function from 1 to 10 in intervals of 0.5.

9.2.a. Define a single-line function to compute the area of a circle given the radius of the circle according to the formula $A = \pi r^2$. Evaluate this function for integer values of 350 to 360.

b. Use a rounding function to display the areas of the circles with the above radii, rounded to the second digit past the decimal place.

9.3 Define a function that computes the length of the hypotenuse of a right triangle given the lengths of the two sides. Evaluate the function with one side equal to 5 while the other has values of 4,3,6,7, and 9.

9.4.a. Define a multiple-line function that converts a number with an octal base to its decimal equivalent. Test your program with the values obtained from the opposite conversion in the program on page 148.

b. What if, in the octal to decimal conversion, the original number has an illegal digit, i.e., a digit greater than or equal to 8? How would you check for an illegal digit and what value would you return for the function?

9.5 Define a multiple-line function to compute the factorial of a non-negative number. Use the function to compute the number of ways that eight books can be arranged on one shelf.

(Method: $P_8^8 = 8\cdot7\cdot6\cdot5\cdot4\cdot3\cdot2\cdot1$)

What if, instead of eight different books, you have only four different books for each of which there are two copies? Determine the number of distinguishable arrangements on one shelf.

$$\text{The number of arrangements} = \frac{8!}{(2!)^4}$$

9.6 Define a multiple-line function to round a numeric value to the hundredths place, and add either a `$` or a `£` before the number. If the fractional part of the number is zero, fill the fractional part of the final number with zeros.

9.7. Write a program that will make it easy for you to manipulate tables of data. First dimension and initialize the elements of an array, then input values for the array elements. Include in your program three subroutines to accomplish the following tasks. Use the subroutines to print or display the sum of the rows and columns of your data table:

1. Write one subroutine to display or print the array.
2. Write a second subroutine that enables you to change a particular array element.
3. Write a third subroutine that finds the sum of each row, the sum of each column and the total sum.

Test your program by finding the row sums, column sums, and total sum of the data in the following table.

12.59	13.69	14.78	?
11.43	22.56	43.78	?
13.52	12.78	14.98	?
?	?	?	

Before you find the row sums, column sums, and total sum of the data in the table, change the value in row 1, column 3 (14.78) to 14.67.

9.8. On his frequent transatlantic missions, chief detective Sylvester S. Py must send encoded messages to the home office. Prior to each mission he supplies the home office with his encoding number. They, in turn, give Sylvester the number they'll use to encode messages sent to him.

Write a BASIC program that uses two subroutines, one to encode messages, the other to decode messages. Use a computed `GOSUB` statement to determine which subroutine is to be accessed. Let the code number be a seed for a sequence of random numbers that encodes the message. (This enables you to use the same random number seed to decode the message.) Use only capital letters in input, coding, or decoding operations. Allow the user to enter one word at a time; you supply the spaces between words.

Suppose Sylvester wants to send the following message to the home office, using his code number (random number seed) .123.

"GET ME TO THE BANK ON TIME"

Run the program to find the encoded message. Run it again, using the same code number to decode the message. Then decode the following message, recently received by Sylvester, using the home office code number .3579.

"NNLSNUNVS IGPXR RQP BVE"

Hint: Use an encoding function like `C$=C$&CHR$(65+(NUM(I$[I,I])+INT(26*RND))MOD 26)` for the length of each word, `I$`.

9.9. Write a "standard pac" program by modifying the row sum and column sum program you wrote for problem 9.7 (sample solution in appendix F) so that the subroutines are performed at the touch of a special function key. Define the special function keys as follows:

`ON KEY #1, "INIT" GOSUB` → Initialize array elements.
`ON KEY #2, "INPUT" GOSUB` → Input values into array.
`ON KEY #3, "COPY-A" GOSUB` → Display or print current array.
`ON KEY #4, "CHANGE" GOSUB` → Change a particular array element.
`ON KEY #5, "SUM" GOSUB` → Sum the rows, columns, and find total sum of array.

We'll leave k6 through k8 for you to define. Additional subroutines might be "ADD," add a row or column to the array; "HELP," display the key definitions; "DELETE," delete a row or column from the array; or "AVG," find the average of the values in a particular row or column.

Run the program to sum the rows and columns of some tables of your own.

Section 10

Printer and Display Formatting

You have seen that the use of commas, semicolons, and quoted text provide limited control of the format of printed or displayed information. Three statements, PRINT USING, DISP USING, and IMAGE, provide the capability of generating printed or displayed output with complete control of the format. The syntax of the statements have two different forms. First, we'll discuss PRINT USING and DISP USING with IMAGE. Later, we'll show that you can specify the format and the information to be formatted in the same statement. Other topics included in this section are:

- Using the TAB function.
- Redefining the printer and the display with the PRINTER IS and CRT IS statements.

Using IMAGE

The IMAGE statement specifies the format by which numbers and strings in the PRINT USING or DISP USING statements will be printed.

PRINT USING statement number [; print using list]
DISP USING statement number [; disp using list]
IMAGE format string

The statement number must refer to an IMAGE statement. The print and disp using lists may be comprised of simple and subscripted variable names, numeric expressions, and string expressions. Functions (including user-defined functions) may be included in the print or disp using list, but if a multiple-line function contains PRINT or DISP statements it may distort the output format. The items in the list are separated by commas or semicolons. However, the commas and semicolons do not affect the format as they do in the PRINT or DISP statements; they merely separate the items in the list. The output is totally controlled by the format string of the IMAGE statement. The format string is a list of field specifiers separated by delimiters. Each field specifier is comprised of special symbols that determine the format of a single item in the print or disp using list. The symbols specify the number of digits, the placement of a comma, decimal point, or blanks—virtually anything having to do with numeric and string output and carriage control.

Each item in the print or display list must correspond to an appropriate numeric or string field specifier.

Delimiters

Two delimiters are used to separate field specifiers:

- , A comma is used only to separate two specifiers.
- / A slash can also be used to separate two specifiers, but its main function is to perform a carriage return and line feed (CR-LF).

The slash, `/`, can be used as a field specifier by itself; that is, it can be separated from other specifiers by a comma. But only the slash delimiter, `/`, can be directly replicated (see page 166).

```
450 PRINT USING 460
460 IMAGE "COST",3/,"DISCOUNT"
COST


DISCOUNT
```

3/ is equivalent to ///.

Prints "COST" and performs 1st CR-LF.
Performs 2nd CR-LF.
Performs 3rd CR-LF.
Prints "DISCOUNT."

The symbols `3/` indicate that three carriage returns and line feeds are to be performed between printing `COST` and `DISCOUNT`. Thus, two blank lines are output.

However, the following image statement would output three blank lines before printing `COST`.

```
460 IMAGE 3/,"COST"



COST
```

Performs 1st CR-LF.
Performs 2nd CR-LF.
Performs 3rd CR-LF.
Prints "COST."

If *n* `/` is at the beginning of an image format string, *n* blank lines are output.
If *n* `/` follows a field specifier in an image format string, *n*–1 blank lines are output.

Blank Spaces

`X` Specifies a blank space.

A number preceding `X` specifies the number of blanks; for instance, `4X` means four blanks. (`XXXX` also specifies four blanks.)

String Specification

Text can be specified in two ways:

`"  "` Text enclosed within quotation marks is printed or displayed exactly as it is quoted. You may specify quoted literals (strings) in either the print or display list or in the `IMAGE` statement.

For example:

```
40 IMAGE "**",4X,"Results",4X,"
   **"
50 PRINT USING 40
**    Results    **
```

`A` Specifies a single string character. A number preceding `A` specifies the number of characters. The length of a string specifier is determined by the number of `A`s that are specified between delimiters; this corresponds to one item in the print/disp using list. When using the `A` string specifier, all text is left-justified.

The above example could also have been written:

```
 90 A$="Results"
100 IMAGE "**",4X,7A,4X,"**"
110 PRINT USING 100 ; A$
**    Results    **
```

7A specifies a field comprised of seven characters. 4X specifies a field comprised of four blanks.

Or like this:

```
130 A$="Results"
140 IMAGE AA,4X,7A,4X,AA
150 PRINT USING 140 ; "**",A$,"*
*"
```

AA can also be represented as 2A.

If the string item in the print/disp using list is longer then the number of characters specified, the string is truncated. For example:

```
180 PRINT USING 190; "RESIDENCE"
190 IMAGE 6A
RESIDE
```

If the item is shorter, the rest of the field is filled with blanks.

Numeric Specification

A variety of symbols can be used to specify numbers: digit symbols, sign symbols, radix symbols, separator symbols, and an exponent symbol.

Digit Symbols

`D` Specifies a digit position. A number preceding `D` specifies the number of digit positions. If the number of `D`s to the left of the decimal point or radix specify a field larger than the numeric item, then the item is right-justified in the field and leading zeros are replaced with spaces. If the number of `D`s to the right of the decimal point or radix specify a field larger than the numeric item, then the item is left-justified in the field with trailing zeros. If the fractional part of the numeric item is larger than the number of `D`s to the right of the decimal point or radix, then the item is rounded to fit the specified field. `D` is the only digit symbol that can be used to specify digits to the right of a decimal point or radix. For example:

```
210 PRINT USING 280 ; 250,25.50
280 IMAGE 5D,2X,DD.DD
  250  25.50
```

`Z` Specifies a digit position—leading zeros are replaced with zeros as a fill character. You cannot use a `Z` to the right of a radix symbol. Again, a number preceding `Z` specifies the number of digit positions. For example:

```
290 PRINT USING 310 ; 256,321
310 IMAGE 5Z,2X,ZZZZZ
00256  00321
```

`*` An asterisk also specifies a digit position, but leading zeros are replaced with asterisks as a fill character. You cannot use an `*` to the right of a radix symbol. A number preceding `*` specifies the number of asterisks. For example:

```
340 IMAGE 5*,2X,5Z,2X,5D
350 PRINT USING 340 ; 99,77,55
***99  00077     55
```

As you can see, any digit symbol, `*`, `Z`, or `D`, can be used to specify the integer portion of any number. But, you cannot mix the symbols in the manner shown below, in the first `IMAGE` statement. For instance, if `D` is used to specify a digit position of a number, all of the number must be specified with `D`'s, except that the digit symbol specifying the one's place can be a `Z` regardless of the other symbols. For example:

```
360 PRINT USING 370 ; 357,972
370 IMAGE DDZZ,2X,D*ZZZ
```

The `IMAGE` statement contains an invalid image and would cause an `Error` message to appear. However, the following image is valid:

```
370 IMAGE DDDZ,2X,*****Z
 357   ***972
```

Radix Symbols

A radix indicator is the symbol that separates the integer part of a number from the fractional part. In the United States, this is customarily the decimal point, as in 34.7. In Europe, this is frequently the comma as in 34,7. One radix symbol at most can appear in a numeric specifier. Only the symbol `D` can be used to specify a digit to the right of the radix indicator.

.	Specifies a decimal point in that position.
`R`	Specifies a comma radix indicator in that position.

Here are some examples:

```
440 PRINT USING 450 ; 473.1,25.3
    92,76.5
450 IMAGE DDD.DD,2X,**Z.DDD,2X,Z
    ZZRDD
473.10  *25.392  076,50

490 IMAGE DDZ.DDD,4X,3Z.3D,4X,Z.
    DD
500 PRINT USING 490 , .756,99.99
    ,.879
  0.756     099.990     0.88
```

Note that .879 is rounded to 0.88 since the image specified only two digits to the right of the radix.

Sign Symbols

Two sign symbols control the output of the sign characters + and −. Only one sign symbol at most can appear in a numeric specifier. When no sign symbol is specified, any minus sign occupies a digit position.

`S`	Specifies output of a sign: + if the number is positive, − if the number is negative.
`M`	Specifies output of a sign: − if the number is negative, a blank if it is positive.

For example:

```
502 PRINT USING 504 ; -47.2,-.51
    ,33.5,38.12
504 IMAGE MDD.DD,2X,SZZ.DD,2X,SZ
    Z.DD,2X,MZZ.DD
-47.20  -00.51  +33.50  38.12
```

The sign ``floats`` with the number; for example:

```
506 PRINT USING 508 ; -5,6,-.07
508 IMAGE SDDD.D,S3D.D,M2D.DD
  -5.0  +6.0  -.07
```

In the examples above, the sign appears immediately to the left of the number. If you use a `Z` or `*` symbol in your format, the minus sign will appear to the left of any leading zeros or asterisks.

Digit Separator Symbols

Digit separators are used to break large numbers into groups of digits (generally three digits per group) for greater readability. In the United States the comma is customarily used; in Europe, the period is commonly used.

`C`	Specifies a comma as a separator in the specified position.
`P`	Specifies a period as a separator in the specified position.

The digit separator symbol is output only if a digit in that item has already been output; the separator must appear between two digits. When leading zeros are generated by the `Z` symbol, they are considered digits and will contain separators. An `IMAGE` format consisting of leading asterisks may contain separators. But if numbers are not output on both sides of the separator, the separator will be replaced with an asterisk.

```
512 PRINT USING 515 ; 25613.92,2
    7.96,71.5
515 IMAGE 3DC3D.DD,2X,ZC3Z.DD,2X
    ,3DC3D.DD
 25,613.92   0,027.96        71.50
```

```
517 DISP USING 520 ; 99,9999.99
520 IMAGE DDD,2X,DCDDD.DD
 99   9,999.99
```

Exponent Symbol

`E`	Specifies that the numeric field that contains `E` is to be output in scientific notation. `E` causes the output of an `E`, the sign of the exponent (+ or −), and a three-digit exponent. At least one digit symbol must precede the `E` symbol.

For example:

```
530 DISP USING 533 ; 157.24
533 IMAGE D.DDDE
1.572E+002
```

```
535 PRINT USING 538 ; 5.762
538 IMAGE DDD.DDE
576.20E-002
```

Compacted Field Specifier

A single symbol, `K`, is used to define an entire field for either a number or a string of characters. If the corresponding print/disp using item is a string, the entire string is output. If it is a numeric, it is output in standard number format (see page 47), except that `K` outputs no leading or trailing blanks. For example:

```
80 PRINT USING 90 ; "ABC",415,"
   DEF",.01
90 IMAGE K,2X,K,K,K
ABC  415DEF.01
```

Replication

Many of the symbols used to make up image specifiers can be repeated to specify multiple symbols by placing an integer in the range 1 through 9999 in front of the symbol. You have already seen some examples; the following `IMAGE` statements, for instance, all specify the same image:

```
540 IMAGE DDD.DD
545 IMAGE D2D.2D
550 IMAGE 3D.DD
555 IMAGE 3D.2D
```

These symbols can be replicated: `D`, `Z`, `X`, `*`, `()`, `A`, and `/`.

In addition to symbol replication an entire specifier or group of specifiers can be replicated by enclosing it in parentheses and placing an integer in the range 1 through 9999 before the parentheses. For example:

```
40 IMAGE DD.D,6(DDD.DD)
50 IMAGE 4Z.D,4(2X,7*Z.D,2(2X,D
   ))
```

So, specifying `3(DD)` is the same as specifying `DD,DD,DD`.

In this manner, `K` can be repeated:

```
60 IMAGE 4(K)
```

Same as specifying `K,K,K,K`.

Up to 128 levels of nested parentheses can be used for replication.

Reusing the IMAGE Format String

A format string is reused from the beginning if it is exhausted before the print using list. For example:

```
150 PRINT USING 155 ; 25.71,99.9
    ,14.23
155 IMAGE DDD.DD
 25.71 99.90 14.23
```

Field Overflow

If a numeric item requires more digits than the field specifier provides, an overflow condition occurs. When this happens, a warning message is displayed and the program continues. For example:

```
160 PRINT USING 165 ; 25.9,336.7
    1,12.1,-14.3
165 IMAGE 4(DD.DD)
Warning 2 on line 160 : OVERFLOW
```

Both numbers 336.71 and –14.3, with an image of `DD.DD`, create an overflow condition. Remember that a minus sign not explicitly specified with `S` or `M` requires a digit position.

Formatting in PRINT/DISP USING Statements

There is another form that a `PRINT USING` or `DISP USING` statement may have, which enables you to specify the image string and the print/disp using list in the same statement:

`PRINT USING` image format string [; print using list]
`DISP USING` image format string [; disp using list]

Instead of specifying the `IMAGE` statement number, you can include the image format string, enclosed within quotation marks in the `PRINT/DISP USING` statements before you specify the print/disp using list. The image format string may be a string enclosed within quotation marks, a string variable, or any string expression that specifies the format.

Examples:

```
 10 PRINT USING "4D DD,2X,**Z.DD
    D"; 1473,25.39
1473.00  *25.390

 20 PRINT USING "3D.2D" ; 310.12
    ,56,42.5
310.12 56.00 42.50

 30 DIM F$[19]
 40 F$="3DC3D.2D,2X,ZC3Z.2D"
 50 DISP USING F$ ; 25613.92,27.
    96
 25,613.92  0,027.96
```

Remember to dimension the string if it is longer than 18 characters.

You *cannot* use quotation marks to specify literal text within an image format string in a `PRINT/DISP USING` statement since quotation marks are used to define the string.

For instance, the following is not allowed and would cause an `Error 84 : EXCESS CHARS` message to appear if you try to enter the statement:

```
50 PRINT USING "NAME",2X,10A,
    "AGE",2D" ; "CHARLES",43
```

The statement is not recognized after the second quotation mark.

An image format string for statement 50 could be specified in either of these ways:

```
50 PRINT USING "4A,2X,10A,3A,3D
   "; "NAME","CHARLES","AGE",43
NAME  CHARLES   AGE 43
```

Or:

```
50 PRINT USING 60 ; "CHARLES",43
60 IMAGE "NAME",2X,10A,"AGE",3D
NAME  CHARLES   AGE 43
```

You can use quoted literals in an `IMAGE` statement since the quotation marks do not define the complete image format string as they do in the `PRINT/DISP USING` statement.

Here is a summary table of image symbols and their uses:

Image Symbol	Symbol Replication	Purpose	Comments
X	Yes	Blank	Can go anywhere
" "	No	Text	Can go anywhere
D	Yes	Digit	Fill=blanks
Z	Yes	Digit	Fill=zeros
*	Yes	Digit	Fill=asterisks
S	No	Sign	"+" or "–"
M	No	Sign	blank or "–"
E	No	Scientific notation	Format=ESDDD
.	No	Radix	Output "."
C	No	Comma	Conditional number separator
R	No	Radix	Output ","
P	No	Decimal point	Conditional number separator
A	Yes	Characters	Strings
( )	Yes	Replicate	For specifiers, not symbols
K	No	Compact	Strings or numerics
,	No	Delimiter	
/	Yes	Delimiter	Output CR-LF

The main factor that must be taken into account with formatted output is the display or printer width. Especially when dealing with numeric output, formatting should be designed so that a line of characters does not exceed the number of characters per line (32 characters per line on the HP-85 printer or display).

The TAB Function

The `TAB` function is used with the `PRINT` and `DISP` statements to print or display information at specified character positions. The main consideration with `TAB` is the length of a line on the printer or display.

`TAB(`character position`)`

The character position may be a number as large as 32767, but you really have 1 through 32 characters positions on either the display or the printer. When the character position specified is greater than the number of columns, it is reduced MOD32.

Example: The following program prints the heading for the variables *X*, *Y*, and *Z*.

```
10 INPUT X,Y,Z
20 PRINT "AVERAGE";TAB(15);"MEA
   N";TAB(26);"MEDIAN"
30 PRINT X;TAB(15);Y;TAB(26);Z
40 END
```

The first heading, `AVERAGE`, starts at character position 0; the heading, `MEAN`, starts at character position 15; and the heading, `MEDIAN`, starts at character position 26. Then in statement 30, the variables are printed under the three headings. If your X, Y, and Z values were input as 11.23, 11, and 11.4, respectively, the printout would be:

```
AVERAGE        MEAN       MEDIAN
 11.23          11         11.4
```

Remember that a comma in a printer or display list outputs the next item in the next print or display zone. Thus, all print or display items used with `TAB` must be separated by semicolons. The `TAB` function cannot be used with the `PRINT USING`, `DISP USING`, or `IMAGE` statements.

Redefining the Printer and the Display

The `PRINTER IS` and the `CRT IS` statements are used to "redefine" the printer and the CRT. Although the statements are most often used with peripherals, you can tell the HP-85 that the display is the printer (`CRT IS 2`); and all display messages from `DISP`, `DISP USING`, `LIST`, Errors, and Warnings will be printed rather than displayed. Or, you can define the printer as the display (`PRINTER IS 1`); all information from `PRINT`, `PRINT USING`, `PLIST`, and `TRACE` statements will be displayed rather than printed. The `PRINTER IS` and `CRT IS` statements are programmable.

`PRINTER IS` output code

`CRT IS` output code

Code	Device
1	CRT
2	PRINTER

For instance, execute:

```
PRINTER IS 1
```

Redefines the printer to be the CRT (display); all `PRINT` statements will be displayed rather than printed.

```
10  PRINT
20  PRINT "**************"
30  PRINT "* NOW CRT IS *"
40  PRINT "*   PRINTER  *"
50  PRINT "**************"
60  GOTO  10
```

Now run the program, the message will be "printed," repeatedly, on the CRT display until you press PAUSE to stop the program.

RUN

```
*   PRINTER  *
**************

**************
* NOW CRT IS *
*   PRINTER  *
**************

**************
* NOW CRT IS *
*   PRINTER  *
**************

**************
* NOW CRT IS *
```

This message will be continuously "scrolled" on the display until you press PAUSE.

PAUSE

After pressing PAUSE, press LIST. Your program will be listed on the display. You can return the system to normal output mode by typing `PRINTER IS 2`, or pressing RESET.

The same can be done with `CRT IS 2` to redefine the CRT. Once `CRT IS 2` is executed, all messages that are normally displayed on the CRT are output to the printer.

For instance:

```
CRT IS 2
DISP "SQR(86)=";SQR(86)
```

These statements cause the following to be printed:

```
SQR(86)= 9.2736184955
```

Again, return the system to normal output mode by executing `CRT IS 1`, or by pressing RESET. The `PRINT ALL` and `COPY` statements are unaffected by the `PRINTER IS` and `CRT IS` statements: `PRINT ALL` and `COPY` always transfer the information from the display to the printer.

Problems

10.1 While considering the variations of social and economic factors among nations of the world, you decide to use the populations, areas, and annual gross national products (GNPs) of various nations to determine their population densities (by dividing the population by the area) and per capita GNPs (by dividing the

GNP by the population). You would like the results to be summarized for each nation. Write a program that requests the name, population, area (in square kilometers), and GNP (in U.S. dollars) of each nation, and prints a summary for each nation according to the following format:

```
  POPULATION     AREA   POP DENS
       ANNUAL GNP      GNP/PERS
--------------------------------
name of nation
 nnn,nnn,nnn   n,nnn,nnn  n,nnn.n
   $n,nnn,nnn,nnn,nnn  $nn,nnn
                :
                :
```

The information below is available for 1977.

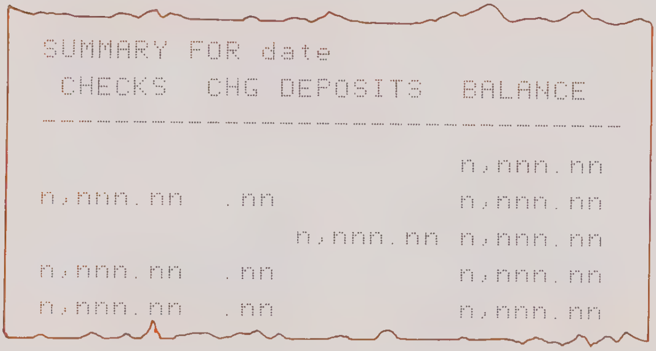
Nation	Population	Area (sq km)	Annual GNP (US $)
China	865,193,550	9,560,980	223,000,000,000
United States	216,817,000	9,363,123	1,781,400,000,000
Canada	23,469,142	9,976,139	195,785,000,000
Singapore	2,322,576	581	5,885,600,000
Mongolia	1,531,940	1,565,000	547,000,000
Qatar	97,792	11,000	4,044,000,000

10.2 In her studies of natural phenomena, physicist Shirley Bright encounters the extremes of length measurement—from the wavelengths of radiation (measured in angstroms) to intergalactic distances (measured in light-years). In order to relate these extremes to each other, Ms. Bright would like to see how a given measurement is expressed in a number of different units, specifically angstroms, meters, and light-years. There are 10^{10} angstroms in a meter and 9.460×10^{15} meters in a light-year. Write a program that converts a measurement (entered as a numerical value and a dimensional unit—A,M, or L) into all three units and prints the three values. An exponential format should be used because of the extremely large and small numbers that are involved. The output should look like:

```
ANGSTROMS   METERS    LIGHT-YEARS
--------------------------------
n.nnnE+nn  n.nnnE+nn  n.nnnE+nn
n.nnnE+nn  n.nnnE+nn  n.nnnE+nn
```

Use this program to express in other units the wave length of light with greatest human visibility (5560 anstroms), the length of the Humber Bridge span in England (1410 meters), the wavelength of certain gamma rays (5.6×10^{-3} angstroms), the approximate diameter of the nucleus of an atom (10^{-14} meters), and the distance to the nearest galaxy (170,000 light-years).

10.3 As an aid in maintaining an accurate record of your checking account, you decide to write a program that takes a sequence of transactions that have occurred over a period of time and prints the status of your account after each transaction. The program is to be initialized by entering the current date and the balance in your account at the beginning of the period. Each deposit is entered as D, *amount*. Each check is entered as C, *amount*. The bank charges 22 cents for processing a check if your balance at that time is less than \$275; there is no charge if your balance is at least \$275. If a check (plus check charge) will overdraw your account, print a negative balance and a special warning giving the amount of the overdraft. Your account summary should have this format:



```
SUMMARY FOR date
 CHECKS   CHG DEPOSITS  BALANCE
--------------------------------
                       n,nnn.nn
n,nnn.nn  .nn          n,nnn.nn
              n,nnn.nn n,nnn.nn
n,nnn.nn  .nn          n,nnn.nn
n,nnn.nn  .nn          n,nnn.nn
```

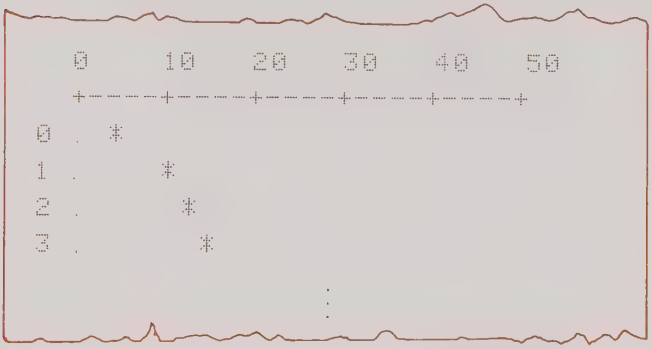
10.4 A regular polygon with n sides inscribed in a circle of diameter d has a perimeter p which is given by

$$p = (d)\,(n)\,\sin\left(\frac{\pi}{n}\right).$$

As the number of sides of the polygon is increased, the polygon more closely resembles a circle, and the ratio of its perimeter to the diameter, p/d, becomes closer to the constant π (which is the ratio of circumference to diameter for a circle). Write a program that lists the perimeter p and the ratio p/d for a series of polygons with n=3,4,5, and so on. Let the diameter d equal 35 units. Have the two columns start at character positions 3 and 19.

10.5 The `TAB` statement may be used to create a graph by varying the character position for each line of output. For example, the data below represents the average weight of a female during her first 18 years. Write a program to produce a printed plot of this data. Each of the 19 years can correspond to a printed line; the position of a printed symbol (such as `#`) can correspond to the weight.

The range of weights, plus the allowance of two spaces to print the age, suggests that the "*" should be printed at the position determined by `TAB(3+W/2)`. It is also helpful to print a "+" at the position of every 10 units of weight across the top of that plot, and a " . " at the position corresponding to zero weight on each line. The plot should resemble the following:



```
  0    10   20   30   40   50
  +----+----+----+----+----+
0 . *
1 .    *
2 .     *
3 .      *
                .
                .
                .
```

Age (years)	Weight (kilograms)
0	3.2
1	9.5
2	11.9
3	13.9
4	15.7
5	17.6
6	19.1
7	21.9
8	24.8
9	28.1
10	32.4
11	37.1
12	41.5
13	46.2
14	50.5
15	53.8
16	55.7
17	56.7
18	56.7

Section 11

Using Tape Cartridges

With your HP-85's built-in tape drive, storing and retrieving programs and data on tape cartridges is convenient and easy. The Getting Started section introduced you to storing programs on magnetic tape. In addition, you can:

- Create data files.
- ''Print'' whole arrays onto the tape with only one program statement—and ''read'' them from the tape just as easily.
- Store an ''Autostart'' program that is automatically loaded and executed at power on.
- Run a large program by storing it in segments on the tape and bringing the parts into computer memory one at a time.
- ''Secure'' program and data files.

The Tape Directory

The tape directory is automatically set up by the computer, providing you with an easily accessible ''table of contents'' of recorded programs and data files. The directory can hold the names of, at most, 42 files. At your request, it directs the system to the exact tape location of recorded programs and data. You need to set up the tape directory only the first time you use a new tape with the HP-85 or whenever you wish to set up a new directory on an old tape whose contents you no longer want.

> **CAUTION**
>
> Do not attempt to remove the cartridge while the tape is in motion or while the tape drive light is on. Damage to the tape and its contents may result.

Directory Set-Up

To set up the tape directory, make sure that the RECORD slide tab is in the right-most position and then initialize the tape with the `ERASETAPE` command. The `ERASETAPE` command renders all previously recorded information on the tape inaccessible.

```
ERASETAPE
```

You must initialize any tape being used for the first time and any recorded tape that is to be erased for re-use. If you execute `CAT` on a tape and a `READ` or `SEARCH` error appears in the display, the tape probably needs to be initialized. (For recurring `READ` errors with a tape that has been initialized, see the Tape Care and Tape Life sections of appendix B.) *When a tape is initialized, all previously recorded information is erased.*

Cataloging

The `CAT` (catalog) command outputs a listing of file names, file types, and physical specifications, enabling you to review the directory contents and to determine the amount of available space remaining on a tape.

```
CAT
```

Place the Standard Pac cartridge into the system's tape transport and execute the `CAT` command. Here is the output you should see:

```
NAME     TYPE     BYTES    RECS  FILE
MOVING   PROG       256      40     1
AMORT    PROG       256      18     2
POLY     PROG       256      29     3
SIMUL    PROG       256      47     4
ROOTS    PROG       256      19     5
CURVE    PROG       256      55     6
FPLOT    PROG       256      22     7
DPLOT    PROG       256      43     8
HISTO    PROG       256      36     9
TEACH    PROG       256      27    10
CALEND   PROG       256      22    11
BIORHY   PROG       256      21    12
TIMER    PROG       256      30    13
COMPZR   PROG       256      56    14
SKI      PROG       256      20    15
MUSIC    DATA       256      44    16
```

Each line of the directory output describes one file in the following manner:

`NAME`	The name given to the file when a program or data is first stored on tape.
`TYPE`	The contents of the file, i.e., `PROG` (program), `DATA`, `NULL` (more on `NULL` later), or `BPGM` (binary program).
`BYTES`	The number of bytes per logical record.
`RECS`	The number of defined records in the file.
`FILE`	The file number assigned by the computer.

Each of the above will be described in the following pages.

Recording and Retrieving Programs

The STORE Command

After a program has been entered into computer memory, executing the `STORE` command attaches a name that you specify to the program, creates a program file, and stores the program on tape in the computer's unique internal language under the name specified.

`STORE` program name

Although the program name is most often specified as a quoted string of at most six characters (e.g., `STORE "PRGM1"`), the name can be any string expression except the null string (e.g., `STORE A$&CHR$(8)`). Longer names are truncated to six characters. Any combination of characters *except* the null string and quotes within quotes (“PRGM“A””) can be used. Remember that spaces are also characters. Because you have the choice of a wide range of character combinations in program names, you may find it advantageous to choose program names that serve to identify the application of the program. For example, a program written to statistically analyze the performance of a business might be named `STAT`, while a program designed for inventory control might be named `INVENT`.

Place a tape cartridge in the tape transport and then enter the following program for arranging numbers in ascending order. If the tape has not been previously used, remember to initialize it with an `ERASETAPE` command. (If you do not wish to do the example program, just key in the first and last program statements and go to page 178.)

```
 10 REM *THIS PROGRAM WILL LIST
    NUMBERS IN INCREASING ORDER
 20 OPTION BASE 1
 30 DIM L(100)
 40 DISP "THIS PROGRAM WILL SORT
     FROM 2 TO 100 NUMBERS IN AS
    CENDING ORDER"
 50 DISP
 60 DISP "HOW MANY NUMBERS ARE T
    O BE SORTED";
 70 INPUT A
 80 A=IP(A)
 90 IF A>100 THEN DISP "TOO MANY
     TO SORT" @ GOTO 60
100 IF A<2 THEN DISP "DON'T BE R
    IDICULOUS" @ GOTO 60
110 DISP
120 DISP "TYPE THE LIST OF NUMBE
    RS ONE AT A TIME."
130 FOR I=1 TO A
140 INPUT L(I)
150 NEXT I
160 DISP
170 DISP "HERE IS THE ORIGINAL L
    IST OF NUMBERS:"
180 FOR J=1 TO A
190 DISP L(J);
200 NEXT J
210 DISP
220 FOR K=1 TO A-1
230 FOR M=K+1 TO A
240 IF L(K)<=L(M) THEN 280
250 R=L(K)
260 L(K)=L(M)
270 L(M)=R
280 NEXT M
290 NEXT K
300 DISP
310 DISP "HERE IS THE LIST ARRAN
    GED IN INCREASING ORDER:"
320 FOR I=1 TO A
330 DISP L(I);
340 NEXT I
350 DISP
360 END
```

Inputs array elements (numbers to be sorted).

Displays original list.

Sorts numbers in ascending order.

Displays sorted list.

Test the sort program by inputting the following list of numbers to see that it arranges them in ascending order:

35, 10, 97, 67, 1, 29, 25, 80, 19

We executed the `PRINT ALL` command before running the program to get the printout below:

```
THIS PROGRAM WILL SORT FROM 2 TO
 100 NUMBERS IN ASCENDING ORDER
HOW MANY NUMBERS ARE TO BE SORTE
D?
9
TYPE THE LIST OF NUMBERS ONE AT
A TIME.
?
35
?
10
?
97
?
67
?
1
?
29
?
25
?
80
?
19
HERE IS THE ORIGINAL LIST OF NUM
BERS:
 35  10  97  67  1  29  25  80
 19
HERE IS THE LIST ARRANGED IN INC
REASING ORDER:
 1  10  19  25  29  35  67  80
 97
```

Now record the sort program on the tape cartridge using the `STORE` command:

```
STORE "SORT1"
```

The above program is now stored on the tape cartridge under the name `SORT1`. (It also remains in computer memory.)

To see how the system has handled `SORT1`, execute `CAT`.

```
CAT
```

Your display should show `SORT1` listed as follows:

```
NAME     TYPE    BYTES    RECS  FILE
  .        .       .        .     .
  .        .       .        .     .
SORT1    PRGM    256        7     *
```

---

* Any other programs you have recorded on the tape will be listed ahead of `SORT1` unless a large enough `NULL` file exists. Most of the time, the file number of a program will match the program's chronological number in the file. (`NULL` files are reused if the program fits.)

The LOAD Command

After a program has been placed on tape using a `STORE` command, it can be retrieved as often as you wish by executing a `LOAD` command.

`LOAD` program name

When retrieving a program with `LOAD`, the program name you use must be the same one assigned to the program when it was recorded with `STORE`. The program name can be a quoted string or any other string expression that specifies the name. The `SORT 1` program you just recorded would be retrieved with the following command:

```
LOAD "SORT1"
```

Executing `LOAD` automatically scratches any program currently in computer memory before loading the new program. The operation also scratches all current data. The `LOAD` command, like the `STORE` command, can be used with any string expression that specifies program name. Just remember that calculator mode variables are scratched when a program is loaded. For instance, to load the `SORT1` program using the string variable `S$`, first assign `SORT1` to `S$`.

```
S$ = "SORT1"
```

Then execute `LOAD` with the `S$` substitution.

```
LOAD S$
```

Now check computer memory by executing `LIST`.

```
LIST
```

When the program is loaded into computer memory with the `LOAD` command, computer memory is scratched; `S$` will no longer be defined.

The CHAIN Statement

The `CHAIN` statement is similar to the `LOAD` command, with four important differences:

- The specified program is executed automatically and immediately as soon as it is loaded into memory.
- As with the `LOAD` command, `CHAIN` destroys any program and data in memory. However, any data stored in common is preserved if the chained program has a `COM` statement.
- The `CHAIN` operation requires that if the program doing the `CHAIN` has a `COM` statement, then the `CHAIN`ed program must have a `COM` statement and the `COM` statements must agree.
- The `CHAIN` statement is programmable.

The syntax for `CHAIN` is simply:

`CHAIN` program name

The chained program must have been previously stored with the `STORE` command. Again, the program name must be a string expression that specifies the program file.

Chaining allows programs of unlimited size to be run by breaking up the program into smaller segments. A `CHAIN` statement in the first segment directs the system to load and run the next segment, preserving variables in common. A `CHAIN` statement in the second segment directs the system to load and run the next segment, preserving variables in common, and so on. (Refer to `COM`, page 123.)

Autostart

An automatic start capability is enabled by storing a program by the name `Autost`:

```
STORE "Autost"
```

You may have noticed that as soon as you turn the power on, the amber tape drive light blinks on and then off again if a tape is in the tape drive. The system is searching for an `Autost` program. At power on, it is instructed to:

```
LOAD "Autost"
```

If a tape cartridge is present in the tape drive at power on and contains a program named `Autost`, that program will be loaded and executed automatically.

The autostart routine permits the computer to load and run a supervisory program automatically, which in turn could define special function keys or chain other programs without operator instructions.

Using Data Files

Along with program storage and retrieval, many computer applications involve the storage, retrieval, use, and updating of data file. You have seen how the `STORE` and `LOAD` commands control the tape-related operations of programming. Because the nature and use of data sets can vary widely, the HP-85 system features five basic data file operations: creating a file, opening a file, recording data, retrieving data, and closing a file. Later in the section, we will discuss the purging, renaming, and securing of files.

Creating a Data File

All data files are initiated with a `CREATE` statement.

`CREATE` file name , number of records [, record length]

The file name must be a quoted string (or any string expression) from one to six characters in length. (The null string is not allowed.) File names serve as identifiers in the `NAME` column of the `CAT` output. For this reason it is best to select file names that best describe the contents of the data files they represent.

Records

Each file you create will be made up of one or more records. A record is the smallest addressable unit on a tape. The number of records you specify for a data file should, in most cases, correspond to the obvious subdivisions in your data. Note that all records in a given file will be of the same length.

Among the features included in your HP-85 is the option to control record length. By specifying record lengths to match your data sets, you can optimize your use of space on the tape cartridges. There are two types of data records:

- **Physical Records** are a standard 256 bytes in length and are *automatically* established with the `STORE` command or whenever [,record length] is omitted from the `CREATE` statement.

```
CREATE "Data 1",4
          ↑      ↑
        File   No. of
        Name   Records
```

  The statement above creates a data file named `Data 1` containing four records. Because no record length was specified in the `CREATE` statement, the records are automatically 256 bytes in length. At most, 850 records of 256 bytes each are allowed in any one `CREATE` statement.

- **Logical Records** can be as small as 4 bytes in length or as large as 32K–1 (32,767) bytes in length, depending on the size you specify in the `CREATE` statement.

```
CREATE "Data 2",4,100
          ↑      ↑  ↑
        File  No. of  No. of Bytes
        Name  Records  per Record
```

The statement above creates a data file named `Data 2` containing four logical records. Because a record length of 100 bytes was specified in the statement, each of the four records is 100 bytes long.

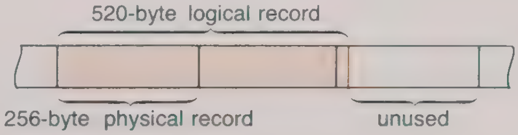
Data Storage

In many cases, logical records will provide the kind of space utilization you want. The following chart description of the number of bytes consumed by numeric and string variables will help you plan logical record sizes for your data files.

Type	Numbers	Strings
Single variable	8 bytes per number	1 byte per character + 3 bytes per string + 3 bytes each time the string crosses into a new defined record.
Array variable	8 bytes X the dimensioned number of elements	None.

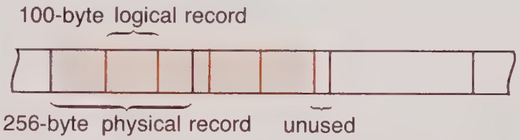
Any *number* of any type (`INTEGER`, `SHORT`, or `REAL`) consumes eight bytes. This means that .1,100, and –49987.532927E499 all consume eight bytes each. All strings consume 1 byte per character.

By summing the number of bytes of storage your data requires, you can tailor your file and define record lengths to suit your needs and minimize waste. However, keep in mind that a file always begins on a new physical record. If you specify a file containing one 520-byte *logical* record (e.g., `CREATE "DATA",1,520`), three 256-byte physical records are required. So 248 bytes are unused, and therefore, wasted space.



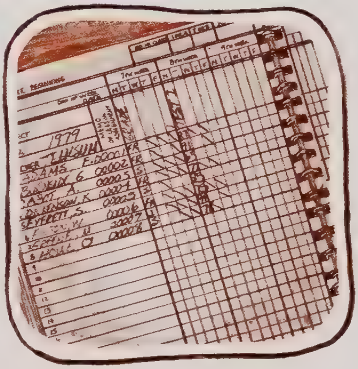
But, whenever you create logical records, the system automatically creates as many logical records as will fit within the physical record space required.

For instance, four 100-byte logical records span two physical records of 256 bytes each. Thus the statement, `CREATE "FILE",4,100`, actually creates five records of 100 bytes each. This way, only 12 bytes are rendered inaccessible rather than 112 bytes.



If a large enough `NULL` file (resulting from a purged file or from a stored program that, after editing, grew too big for its original file) exists on the tape when you create a file, the `NULL` file will be reused. The system will also completely fill the `NULL` file with logical records of length specified in the `CREATE` statement. For instance, if a `NULL` file containing six physical records (1536 bytes) exists on the tape when you execute `CREATE "FILE",4,100`, a total of 15 records of 100 bytes each will be created. To avoid wasting space on the tape, you may wish to store a "DUMMY" program in the `NULL` file until a larger program or file can be stored there.

Example: In a class record book, Professor I. Tehlsum has entered the name, student number, year, and first test score of each of eight computer students. If Tehlsum also wants to make a simple record of the data on an HP data cartridge, how much space will be required?



Name	Student No.	Year	Score
Adams E	00001	Frsh	87
Burnside G	00002	Frsh	91
Cabot A	00003	Soph	63
Dickenson K	00004	Frsh	77
Everett S	00005	Soph	85
Fulton W	00006	Frsh	75
Greene M	00007	Junr	98
Howe O	00008	Soph	78

The student names, numbers, and years each represent separate strings. (Student numbers are stored as strings in order to preserve leading zeros, e.g., `"00001"`.) Each score is a separate numeric variable. How many physical records will Tehlsum's student data require? The professor quickly totals the approximate data space needed as follows:

8 names, allowing 12 characters per name	= 96 bytes
+ 3 bytes for each string.	= 24
8 student numbers of 5 bytes each	= 40
+ 3 bytes for each string.	= 24
8 "year" strings of 4 bytes each	= 32
+ 3 bytes for each string.	= 24
8 numeric variables (scores) of 8 bytes each	= 64
Total bytes	304

The student data requires approximately 304 bytes. As 256 bytes represents the maximum a single physical record can hold, either two physical records of 256 bytes each, or one logical record of 304 bytes will be required.

Place a tape cartridge in your HP-85's tape drive and create a file for Professor Tehlsum's student data.

```
CREATE "CLASS1",1,304
```

File Name
No. of Records
Record Length

The result of the above operation is a data file named `CLASS1`, composed of one logical record of 304 bytes. The directory now contains a record of the data file you have created. Execute `CAT` to display:

```
CAT
NAME      TYPE     BYTES     RECS  FILE
CLASS1    DATA       304        1     1*
```

When you create data files, be sure that the length and number of your physical and logical records suit the storage requirements of your data. If, in the example above, you had specified one record of 250 bytes instead of 304 bytes (`CREATE "CLASS1",1,250`), an `EOF` (end-of-file), `RANDOM OVF`, or `RECORD#` error would occur if you attempted to record the data. `EOF` errors result whenever an attempt is made to store more bytes of data than the file has been specified to handle.

Opening a Data File

Even though a data file has been established in the directory with a `CREATE` statement, it cannot be accessed for data storage until it is "opened" using the `ASSIGN#` statement.

`ASSIGN#` buffer number `TO` file name

The buffer number is a number from 1 through 10. The file name must be a quoted string or any other string expression that specifies the name of a previously created file. To access any data file, whether just created or previously established, the file must be opened by assigning it to a buffer.

Buffers reduce tape wear and increase efficiency by reducing the number of transfers to the tape. They function "behind the scenes" in your HP-85 computer, so you don't have to be too concerned about how they work. However, it may be helpful to know that data you wish to record on tape is first collected in a 256-byte buffer. The buffer is allocated in computer memory when you assign it to a file name; it returns to regular read/write memory (available to the user) when you're through using it for buffering data. The HP-85 can reserve at most 10 data buffers at any time.

`PRINT#` statements (see page 185) cause data transfers to the buffer (rather than to the tape); each time the buffer's contents reach 256 bytes of data, it automatically records the data onto the tape. You'll know when the buffer records or retrieves data from the tape: the display will blink off and the tape drive light will blink on as the tape is being accessed.

---

* (`CLASS1` will be file 1 if no other files precede it on the tape.)

A buffer that is assigned to a file name is also recorded under these conditions:

- Assigning that buffer number to a different file.
- Executing a `PAUSE`, PAUSE, `STOP`, or `END`.
- Pressing any key that interrupts program execution.
- Closing the file.
- Random access to another logical record (we'll discuss this shortly).
- Executing a `PRINT#` statement from the keyboard.

If you wish to have more than one file open at a time, simply add to your instructions a separate `ASSIGN` statement and buffer number for each additional file. Just remember that assigning a buffer currently in use to a new file "closes" the current file and reassigns the buffer to the new file.

Examples:

```
150 ASSIGN# 1 TO "CLASS1"
160 ASSIGN# 4 TO "Data12"
170 ASSIGN# 10 TO "NUMS"
```

Assigns buffer 1 to `CLASS1`.
Assigns buffer 4 to `Data12`.
Assigns buffer 10 to `NUMS`.

Buffer No.

File Name

Once you have created a file and opened it (by assigning a buffer to it), you can record data in the file or read data from a previously recorded file.

Closing a Data File

When you finish recording data into a file, always close the file with a special `ASSIGN` statement before proceeding with other operations.

`ASSIGN#` buffer number `TO *`

The `ASSIGN` statement used in closing a file is always identical to the `ASSIGN` statement used to open the file, *except* the file name is replaced by an asterisk. For example, the `ASSIGN` statement we will use later to close the student data file will appear as follows:

```
ASSIGN# 1 TO *
```

Buffer Number

Remember from our section on *buffers* that some information is kept in memory instead of being immediately stored on the tape. If a program error causes a halt when some information is in the memory buffer but not yet on tape, that part of the data in *memory only* will be lost unless you *close* the file. When you close a file (`ASSIGN#` buffer number `TO *`) all information in the buffer is recorded on the tape and the buffer is released. Since the buffering process is "invisible," it is important to close all files after use, even when errors occur.

Storing and Retrieving Data

A data file opened by an `ASSIGN#` statement is ready for data access. There are two methods for both data storage and data retrieval operations: **serial file access** and **random file access**. Your choice of access depends on the way in which your data is to be used.

Serial File Access

Serial file access is normally used to record or retrieve masses of data items sequentially, without regard to logical records. Your use of serial file access should be planned for applications requiring this non-selective form of recording and retrieval. Collections of string or numeric data that will be recorded or retrieved sequentially as a complete list are the best candidates for serial file access.

When data is recorded onto the tape serially, it can be longer or shorter than the logical record length. (For example, a long string might span three logical records.) For each data file opened, a file pointer keeps track of the data item currently being accessed. As you store or retrieve data, the pointer *moves serially forward* through the file.

Writing Serial Files

The serial `PRINT#` statement records values into the specified file from the variables, strings, or numbers in computer memory without regard to record subdivisions.

`PRINT#` **buffer number** `;` **print # list**

The buffer number (1-10) used in the `PRINT#` statement must be the same buffer number you have assigned to the file with the `ASSIGN#` statement.

The print# list is a sequential listing of data items identified for writing (recording) on tape. Items in the print# list are separated by commas and can be numbers, variables, strings, or array names.

If nothing has been stored or retrieved, writing begins at the beginning of the file specified. Otherwise, writing begins immediately after the data item most recently stored or retrieved.

Example:

```
110 FOR S=1  TO 8
120 INPUT  A$,B$,C$,D
130 PRINT# 1 ;  A$,B$,C$,D
140 NEXT S
```

These statements record values for A$, B$, C$, and D onto a tape file through buffer #1. The file pointer remains at the end of the data list until:

- A `PRINT#` statement is executed, writing data to the same file.
- The file is closed.
- A `READ#` statement is executed (more on `READ#` shortly).

Advanced Programming Note: When storing a long *string*, it might be too long to be contained in one logical record. As long as you have already allocated enough records in the file, the string is automatically broken up into as many pieces as logical records that are needed. This adds *three* bytes for each time the string crosses over into another logical record. These bytes compose the string "header," which identifies the parts of the string as first, intermediate, or last, and specifies the length of each part.

> But, when insufficient space remains in a nearly full record to completely write all eight bytes of a *number,* the number will be written entirely in the next record. The remaining bytes in the first record are unused, resulting in from one to seven bytes of wasted space. This happens because all numbers require eight bytes of storage space on the tape, and cannot be split between two records.
>
> For example, if a record has only five bytes remaining in which to write a number, the five will be skipped and the number will be written in the first eight bytes of the following record. Note that crossing over into a new record to write a number does not cause a header to be written. Only string data, which can be split between two or more records, require headers.

When you set up a serial file of two or more records, avoid an unexpected space shortage by allowing some extra bytes in the record length. This will compensate for any additional bytes that might be consumed by strings or numbers crossing record boundaries.

The length of the data in the list must be less than or equal to the storage space that remains in the file after the pointer; otherwise, an EOF (end-of-file) error occurs, signaling that you have filled your file.

Example: By following the steps covered in the preceding pages, write a program to create a file for Professor Tehlsum's student data, assign a buffer to the file, and record the information from the table on page 182 onto the tape. (Do not use a `CREATE` statement in the program if you have already created the file.)

```
 10 CREATE "CLASS1",1,304                  Creates the file.
 20 ASSIGN# 1 TO "CLASS1"                  Opens the file.
 30 DISP "TYPE IN STUDENT NAME,            User instruction.
    NUMBER, YEAR, AND SCORE."
 40 FOR S=1 TO 8
 50 INPUT A$,B$,C$,D                       Inputs data.
 60 PRINT# 1 ; A$,B$,C$,D                  Records data in buffer assigned to the file.
 70 NEXT S
 80 ASSIGN# 1 TO *                         Closes the file.
 90 DISP "END"                             Signals end of program.
100 END                                    Halts program.
```

Now run the program to input the student information and record it in a data file.

```
RUN
TYPE IN STUDENT NAME, NUMBER, YE
AR, AND SCORE.
?
ADAMS E, 00001, FRSH, 87
?
BURNSIDE G, 00002, FRSH, 91
?
CABOT A, 00003, SOPH, 63
?
DICKENSON K, 00004, FRSH, 77
?
EVERETT S, 00005, SOPH, 85
?
FULTON W, 00006, FRSH, 75
?
GREENE M, 00007, JUNR, 98
?
HOWE O, 00008, SOPH, 78
END
```

The student data is now recorded. At this point the recorded tape can be removed from the tape drive and stored for future use. Now let's discuss the serial READ# statement to retrieve the data you've recorded.

Reading Serial Files

Before you can use data that has been stored in a data file with a PRINT# statement, you must read the data back into computer memory with a READ# statement. Data is merely copied into computer memory.

Using the serial READ# statement, you can read numbers, strings, or array values from recorded files into computer memory in the same way that they were recorded using a PRINT# statement.

READ# buffer number ; read list

The read list, like the print list, is the sequential listing of data items identified for reading from the tape. Items in the read list are separated by commas. The variables in the read list do not have to have the same names as specified in the PRINT# statement. But, the variable names in the read list must match in number and type (string vs. numeric) the PRINT# statement(s) print list previously stored. If the READ# statement(s) specify more data items than were originally stored, an EOR (or EOF) error occurs, meaning there is no more data.

Remember, data read must correspond to the type—numeric or string—that was printed. However, a numeric item need not be of the same precision (REAL, INTEGER, or SHORT). Precision is automatically converted.

As we shall see, you can also record an entire array and read it back as simple variables or as other arrays, and vice versa.

To begin reading from the beginning of the file, you must reposition the pointer (using a random READ# statement, as you will see later) or do another ASSIGN. Data can be updated and restored into the file or into a new file.

Example: Write a program (or modify the last program, page 186) to read Professor Tehlsum's student data from the CLASS1 file and display it on the CRT.

Note: You do not need to use another CREATE statement; the file already exists on the tape.

```
 20 ASSIGN# 1 TO "CLASS1"
```

Opens the file (assigns a buffer to it).

```
 30 DISP "HERE ARE STUDENT NAMES
    , NUMBERS, YEARS, AND SCORES
    ."
```

User instructions.

```
 40 FOR S=1 TO 8
 60 READ# 1 ; S$,N$,Y$,T1
```

Variable names do not need to be the same as in the PRINT# statement but they must agree in number and type (numeric or string).

```
 65 DISP S$,N$,Y$,T1
 70 NEXT S
 80 ASSIGN# 1 TO *
```

Closes the file.

```
 90 DISP "END"
100 END
```

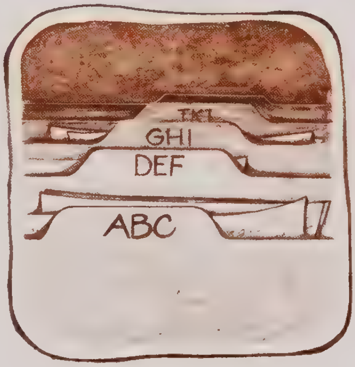
Now run the program to display the information that was previously printed in the data file.

```
RUN
HERE ARE STUDENT NAMES, NUMBERS,
YEARS, AND SCORES.
ADAMS E            00001
FRSH                87
BURNSIDE G         00002
FRSH                91
CABOT A            00003
SOPH                63
DICKENSON K        00004
FRSH                77
EVERETT S          00005
SOPH                85
FULTON W           00006
FRSH                75
GREENE M           00007
JUNR                98
HOWE O             00008
SOPH                78
END
```

Note that in serial `READ#` operations the data pointer moves in the same manner as it does in serial `PRINT#` operations, marking in computer memory where the last data item ended and where the next data item should begin.

Random File Access

Random file access is used to store or retrieve data items from a specific logical record, thus, the name—you can access any record at random. Random file access requires you to specify the logical record that you wish to access. The file pointer is positioned at the beginning of that record.



You should use random file access methods when you wish to be able to store or retrieve groups of data items from specific logical records by referring to the record number.

Random Writing

The random `PRINT#` statement is nearly identical to the serial `PRINT#` statement except that:

- The record number must be specified.
- Data is recorded into the file starting at the beginning of the specified record.
- End of record marks are not ignored; thus, data cannot be larger than the logical record length (e.g., a long string must fit entirely within the record specified with random file access).

`PRINT#` buffer number `,` record number [`;` [print# list]]

The print# list is identical to that used in the serial `PRINT#` statement. The random `PRINT#` statement records data into the specified record of the file. With random file access, printing always starts at the beginning of the specified logical record. Any previous data is overwritten. Any data not overwritten because the new logical record is shorter is rendered inaccessible by the new end-of-record mark.

Again, the print# data list must fit in the logical record or else an EOR error occurs. If you attempt to specify a logical record number greater than the number of records specified in the `CREATE` statement, an EOF error occurs.

When no print# list is specified and a semicolon follows the record number, the data pointer is repositioned to the beginning of the specified record.

Examples:

```
170 PRINT# 3,1 ; A,B
180 PRINT# 3,2 ; C,D
190 PRINT# 3,3 ;
```

Records A and B in record 1.
Records C and D in record 2.
Repositions the pointer to the beginning of record 3.

Example: Write a program that creates a new file for Professor Tehlsum's student data so that information pertaining to a particular student may be accessed at random; i.e., create one record for each student. Then read the information from the serial file that you created (page 182) and write it into the new file.

```
 10 CREATE "STUDE1",8,40


 20 ASSIGN# 1 TO "STUDE1"
 30 ASSIGN# 2 TO "CLASS1"

 40 FOR I=1 TO 8
 50 READ# 2 ; S$,N$,Y$,T1
 60 PRINT# 1,I ; S$,N$,Y$,T1
 70 NEXT I
 80 ASSIGN# 1 TO *
 90 ASSIGN# 2 TO *
100 DISP "END"
110 END
```

Creates a file of 8 records, 40 bytes each. (The system actually creates 12 records to span two physical records.)
Opens the file.
Opens previously recorded file (see program, page 186).

Reads data from old file into buffer 2.
Writes data into specified record.

Closes `STUDE1` file.
Closes `CLASS1` file.
Signifies end of program.

In the program above, when I = 1, statement 60 writes the values of the variables into the first record:

```
PRINT# 1,1 ; S,N$,Y$,T1
```

When I = 2, statement 60 writes the values of the variables into the second record:

```
PRINT# 1,2 ; S$,N$,Y$,T1
```

And so on for the remaining six records.

Random Reading

The random `READ#` statement is like the serial `READ#` statement except that reading of data into computer memory starts at the beginning of the specified logical record and will not read past an EOR or EOF mark. It is called "random `READ#`" because you can read a particular record at random.

`READ#` buffer number `,` record number [`;` variable list]

Again, as in the serial `READ#` statement, the variables into which you read values do not necessarily have to have the same names, precision, or type (`INTEGER`, `REAL`, or `SHORT`) as specified in the `PRINT#` statement. But the variable names must match the data as to string or numeric type.

If the number of items making up the variable list is greater than the data in the logical record, an EOR error occurs.

Example: Now professor Tehlsum can easily access and change or update the data in each student's record. Write a program that will read data from a particular record of your choosing, display the data, and accept changes. Then incorporate the changes by rewriting the record.

```
  10 ASSIGN# 1 TO "STUDE1"
  20 DISP "WHICH RECORD DO YOU
     WISH TO VIEW (1-8)";
  30 INPUT R
• 40 READ# 1,R ; S$,N$,Y$,T1
  50 DISP S$,N$,Y$,T1
  60 DISP "ANY CHANGES (Y/N)";
  70 INPUT A$
  80 IF A$="N" THEN 120
  90 DISP "ENTER NEW TEST SCORE"
 100 INPUT A1
•110 PRINT# 1,R ; S$,N$,Y$,A1
 120 DISP "DO YOU WISH TO VIEW AN
     OTHER RECORD (Y/N)";
 130 INPUT A$
 140 IF A$="Y" THEN 20
 150 ASSIGN# 1 TO *
 160 DISP "END CHANGES"
 170 END
```

Opens the file.

Reads data from record.
Displays contents of record.

Inputs new score.
Rewrites record with change.

Closes the file.

Repositioning the Pointer

If the semicolon and variable list are omitted from the random `READ#` statement or from the random `PRINT#` statement, the file pointer is repositioned to the beginning of the specified record. To reposition the pointer to the beginning of a file (e.g., for use with serial file access) execute:

`READ#` buffer number `, 1`
`PRINT#` buffer number `, 1`
`PRINT#` buffer number `, 1 ;`

All statements reposition the file pointer to the beginning of the specified record—in this case, record 1.

Storing and Retrieving Arrays

Entire arrays can be stored and retrieved by using the following notation with the `PRINT#` and `READ#` statements with serial *or* random file access:

One-dimensional arrays: array name `()`.
e.g., `L()`, `X5()`, `T()`.
Two-dimensional arrays: array name `(,)`.
e.g., `L(,)`, `X5(,)`, `T(,)`.

Most of the time, a comma is used for documentation purposes. If an array has been dimensioned previously, its use is optional.

Arrays are stored and retrieved element by element without regard to dimensionality with the last subscript varying the fastest (in other words, by rows).

Example: The following program reads test scores and averages into array T, then, with one program statement, writes the entire array into one record.

```
 10 REM *SCORES AND AVES*
 20 OPTION BASE 1
 30 DIM T(3,5)
 40 FOR I=1 TO 3
 50 READ T(I,1),T(I,2),T(I,3),T(
    I,4)
 60 T(I,5)=(T(I,1)+T(I,2)+T(I,3)
    +T(I,4))/4
 70 NEXT I
 80 CREATE "TESTS",4,128

 90 ASSIGN# 1 TO "TESTS"
•100 PRINT# 1,2 ; T(,)
110 ASSIGN# 1 TO *
120 DISP "Array of test scores a
    re now recorded in RECORD 2"
130 DATA 78,43,69,81,98,99,92,97
    ,55,50,75,72
140 END
```

Specifies lower bound 1 in arrays.
Dimensions array T, 3×5.
Reads array elements.
Finds average.
Creates file of 4 records, 128 bytes per record.
Opens file (assigns buffer).
Writes entire array in record 2.
Closes file (dumps buffer).

Now you can retrieve the data in record 2 in the same manner:

```
160 ASSIGN# 1 TO "TESTS"
•170 READ# 1,2 ; T(,)
180 DISP "AVERAGES"
190 FOR I=1 TO 3
200 DISP T(I,5);
210 NEXT I
220 DISP
230 END
```

Reads entire array from the file.

If you are appending this to the previous program, be sure to delete statement 140. (Also, delete statement 80 if you have created the file previously.)

Or you can read the data in record 2 of `TESTS` back into simple variables serially; but first you must reposition the pointer.

```
170 READ# 1,2

180 READ# 1; A,B,C,D,E
190 DISP A,B,C,D,E
```

Positions the pointer at the beginning of record 2.
Reads first five items.
Displays test scores and average in first row of array.

Purging a File

The `PURGE` statement prevents access to any file (program, data, etc.) by removing its name from the name column in the tape directory. `PURGE` makes the specified file available for the storage of new programs or data.

`PURGE` "file name" [, purge code]

The file name must be the name of the file you wish to purge; the same name that is listed in the directory. If no purge code is specified, the system purges the particular file specified by returning the records of the file to "available storage space" on the tape. This is indicated by the word `NULL` in the type column of the directory. For example:

```
PURGE "DATA1"
```

Renders the previously created file inaccessible by removing its name in the directory.

Any future program or data file that fits will be stored or created in the first available `NULL` file; otherwise it is stored at the end of the list of previously stored programs or data files. Regardless of how small the program or data file is, if a large enough `NULL` file exists, the program or data file will be created within the `NULL` file. The directory will assume that you have used the entire `NULL` file to store the program (or create the data file, as we have seen earlier).

Sometimes it may be desirable to purge the tape *from a specific file to the end of the tape*. If, for instance, you eject the tape when it was in the process of recording information onto a file, that file and the rest of the tape might be rendered inaccessible. In order to make that part of the tape available for storage once again, specify a purge code of 0 following the file name. For example:

```
PURGE "DATA1",0
```

Returns `DATA1` and any programs or data files that follow it on the tape to "available storage space."

A purge code of 0 removes the names of all files following and including the specified file from the directory. A `NULL` file marker is not written into the directory; the system will assume that you've only recorded programs or data up to the first purged file.

Any number, other than zero, used with `PURGE` operates in the same manner as specifying `PURGE` without a purge code; the specified file only is purged from the tape directory.

Renaming a File

The `RENAME` statement is used to give a file a different name.

`RENAME` old file name `TO` new file name

For example:

```
RENAME "TESTS" TO "TERM1"
```

Any file may be given a new name, including data files and secured files, as long as a name for the file exists in the directory.

Binary Programs

Some of the programs or routines in the application pacs are binary programs. A binary program is equivalent to a plug-in ROM except that you load it from a tape cartridge. Although binary programs will come with instructions about their use, for now, note that you can load or store binary programs with the `LOADBIN` and `STOREBIN` statements.

`STOREBIN` program name

A binary routine is retrieved and added to the current program in memory using the `LOADBIN` statement.

`LOADBIN` program name

The program name for both statements may be a quoted string from one through six characters in length or it may be any string expression specifying the program name. For example:

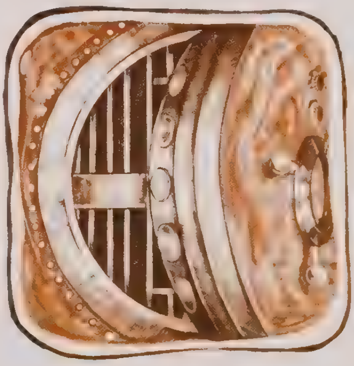
`100 STOREBIN "ROOTS"`	Stores binary program named `ROOTS`.
`200 LOADBIN "MUSIC"`	Loads binary program named `MUSIC`, without altering any existing program or data in memory.

One binary program, at most, may be in computer memory at any time. Remember that in order to edit a program that uses a binary routine, that binary routine must be present in computer memory.

Securing Files

File security enables you to prevent your programs or data from being copied or changed, or accidently overwritten.

The `SECURE` command is used to prevent specific program files from being listed, edited, or stored, to prevent any file's name from appearing in the directory listing, and to protect the user from writing over a file. The `UNSECURE` command is used to remove security on secured programs or data files.



`SECURE` file name , security code , secure type

The file name that is being secured must already be listed in the directory (i.e., the program or file being secured must already exist on the tape). It may be specified as either a quoted string or as any other string expression that specifies the file. The security code may be any string of characters except the null string. The system takes the first two characters of the string specified and stores them as the security code. If only one character is specified, the second character is a blank.

The security type is specified by a number from 0 through 3. Possible security types are:

Number	Secured Against
0	LIST, PLIST, and Editing
1	LIST, PLIST, Editing, and STORE (duplication)
2	STORE (overwriting), PRINT#, STOREBIN
3	CAT (a blank appears where name should be)

Security types 0 and 1 can be used with program files only. Type 2 can be used with any file. Type 3 is intended for data file security but it may be used with program files.

Examples:

`SECURE "SPECS","DD",1`	Secures program `SPECS` so that it cannot be edited, stored, or listed.
`SECURE "DATA18","18C",3`	Secures data file `DATA18` so that in place of the name, blanks appear in the directory listing. The security code is `18`, the first two characters of the string.

The `SECURE` command is programmable. You can secure a program file with more than one security type. For instance, you can secure a program file against listing, editing, and storing (duplicating) or secure its name from being listed in the directory or both. The only types you cannot secure at the same time are types 0 and 1.

The difference between `STORE` in types 1 and 2 is that a type 1 security prevents the program from being duplicated or stored in another file. Type 2 security protects a file from being over-written. If you attempt to store a program using the same name as a file secured with type 2 security, a warning that the file is write-protected will be output. Type 2 security protects against `STORE`, `STORE BIN`, and `PRINT#` only; you can always purge the file regardless of its security.

The `UNSECURE` command operates in the same manner as the `SECURE` command to remove the security on specified files. But remember you *cannot* unsecure an open data file. In other words, the `SECURE` and `UNSECURE` commands affect the tape file and the directory only—not the memory buffer. If you attempt to unsecure an open data file, you'll probably get a `SECURE` error since the buffer will register that the file is secured.

`UNSECURE` file name `,` security code `,` secure type

The file name must be that of a secured file and the security code types 0 and 1 must have the same first two characters as the code that was originally used to secure the file. You can use any two characters as the security code for unsecuring types 2 and 3.

Examples:

```
UNSECURE "SPECS","DD",1
```

Unsecures program `SPECS` so that it can be stored, listed, and edited.

```
UNSECURE "DATA18","KH",3
```

Unsecures data file `DATA18`. Note that the security code need not match the original security code for types 2 and 3. Now the data file name will appear in a directory listing.

The `UNSECURE` command is not programmable.

The HP-85 provides two more tape operations: `REWIND` and `CTAPE` (condition tape). They are discussed in appendix B.

NO. % NET WEIGHT DISTRIBUTION
7 .28
6 .24
5 .20
4 .16
3 .12
2 .08
1 .04
0 0
LIM:
165
170
175
180

Section 12

Graphics

The graphics capabilities of your HP-85 truly enhance your BASIC programming power. HP-85 graphics enable you to:

- Plot data on the graphics display, thus clarifying complex sets of information in pictorial form.
- Scale the display yourself to desired proportions.
- Generate an unlimited number of lines, curves, diagrams, and designs on the display.
- Copy anything from the graphics display to the printer with one command.
- ''Draw'' and label graphs with ease.
- Interact with the graphics display from the keyboard.
- Execute any of the graphics commands from the keyboard or in a program.

The Graphics Display

The HP-85 provides two different display areas or modes: *alphanumeric* and *graphics*. Normally the display is in *alpha* mode, but you can view the current graphics display at any time by pressing the GRAPH key or by executing the statement, `GRAPH`.

```
GRAPH
```

Sets the display to graphics mode.

Any of the graphics statements that directly manipulate the graphics display also set the display to graphics mode automatically. You can return to *alpha* mode by pressing any alphanumeric key or display control key (such as the space bar or the ↑ key) or by executing the `ALPHA` statement:

```
ALPHA
```

Sets the display to alpha mode.

To get an idea of the graphics display area available for your use, enter the following program into the computer and then press RUN. This program will frame (draw a box around) the CRT graphics display area that is available to you.

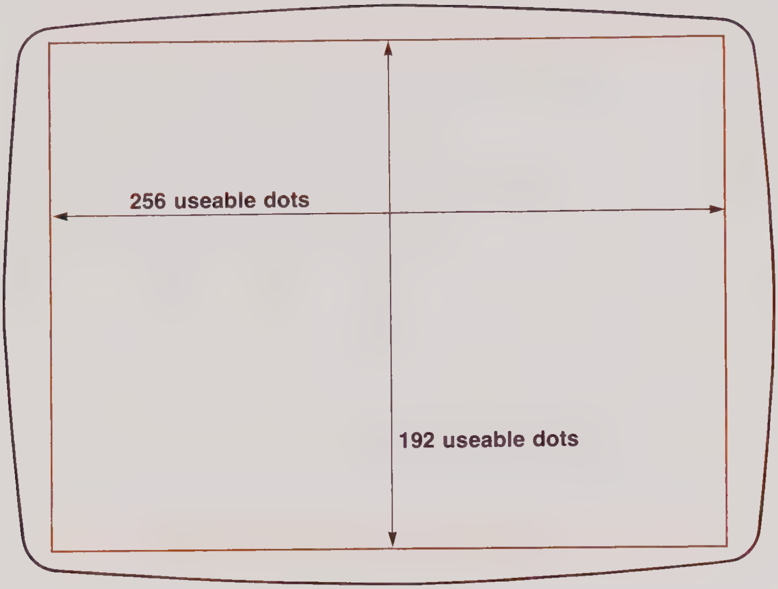
First, press SCRATCH to clear the system memory of previous programs.

```
10 GCLEAR
20 SCALE 0,100,0,100
30 XAXIS 0 @ XAXIS 100
40 YAXIS 0 @ YAXIS 100
50 END
```

Clears the graphics display.
Scales the graphics display.
Draws a frame around the plotting area.

This example program (and others like it found throughout this section) is given to provide some hands-on experience with HP-85 graphics and to illustrate various statements. Each of the graphics statements will be explained at appropriate places in the section.

When you run the program, the display shows:



This frames the graphics display area. You have 256 useable dots in the horizontal direction and 192 useable dots in the vertical direction, yielding a total of 49,152 points available for plotting.

By useable dots, we mean the actual physical dots of the graphics display screen. As you shall see, the display may be scaled to horizontal and vertical units of your own choosing. Points are plotted according to the current scale; they are automatically *mapped* onto the graphics display screen.

Line Generation

Line generation refers to the process of producing a line on the graphics display, which is similar to drawing a line with a pen. But the display has no actual pen. The display *does* have a point, referred to as "the pen," which when moved produces a line (or row of dots) if line generation is turned on *(pen down)*. If line generation is turned off *(pen up)* no line is produced, but the point moves.

Graphics and the Printer

The general method of performing HP-85 graphics is:

1. First generate your graph or design on the graphics display using the graphics statements either from the keyboard or within a program.
2. Then, to produce a hard copy of your graphics, simply set the display to graphics mode by pressing GRAPH and then press COPY. In a program, these same operations can be performed by executing the `GRAPH` statement followed by the `COPY` statement. (`GRAPH` need not be executed if the display is already in graphics mode.)

The printer generates the graphics display sideways to assure that it fits properly on the paper and to enable strip charting.

Clearing the Graphics Display

The `GCLEAR` statement clears the graphics display of any previously plotted data.

`GCLEAR` [Y-coordinate]

The `GCLEAR` statement clears the graphics screen from the specified Y-value to the bottom of the screen. For instance, if the graphics display is scaled from 0 to 100 in the vertical direction, execute the following to clear the lower half of the display:

```
GCLEAR 50
```

Clears lower half of graphics display with vertical scale of 0 to 100.

If no parameter is specified, `GCLEAR` clears the entire graphics screen.

It is advisable to use the `GCLEAR` statement before you begin a new plot in a program, thus assuring that you do not plot over any previous graphics.

Execute `GCLEAR` now to clear the frame from our first graphics program. The display will change to alpha mode when you type in a graphics statement. It reverts back to graphics mode to show the change in the graphics display once the command is executed. The `GCLEAR` statement clears the graphic display to the current background "color" (more about this later).

Setting Up the Graphics Display

A program written to plot or draw lines on the graphics display usually includes some initial set-up operations to define the plotting area. Typical set-up operations might be clearing the display and framing it, as we did earlier. Most often, the display is *scaled* to the desired proportions before any plotting is done.

For instance, you might use the following group of statements to set up the graphics display.

```
10 GCLEAR
20 SCALE -10,10,-10,10
30 XAXIS 0,1
40 YAXIS 0,1
50 END
```

These statements will be discussed in the following pages.

The SCALE Statement

The `SCALE` statement defines the minimum and maximum values of the X (horizontal) and Y (vertical) directions for the graphics display. This enables you to specify your own units for plotting.

`SCALE` Xmin , Xmax , Ymin , Ymax

The first two parameters specify the values represented by the left and right boundaries of the graphics display. The last two parameters specify the values represented by the lower and upper boundaries of the display. If Xmax is less than Xmin or Ymax is less than Ymin, an error occurs.

At power on or after pressing RESET, the minimum and maximum values of both X and Y directions are 0 and 100:

```
SCALE 0,100,0,100
```

Specifies X and Y units from 0 to 100.

The `SCALE` statement may be used to place the origin (point 0,0) on or off the graphics display.

For example, if you want to plot the average annual rainfall at a weather station for a 10-year period, the `SCALE` statement might look like this:

```
SCALE 1968,1978,0,20
```

The left edge of the graphics display area would represent the year 1968 and the right edge would represent 1978. Rainfall would be plotted in the Y direction in volume units (e.g., inches). This enables you to plot data in years and volume units (e.g., point 1976, 7) directly on the graphics display area.

More Examples:

```
SCALE 0,10,0,10
SCALE -30,20,-10,20
```

Scales X and Y from 0 to 10.
Scales the graphics display 50 X-units wide and 30 Y-units high.

Unequal Unit Scaling

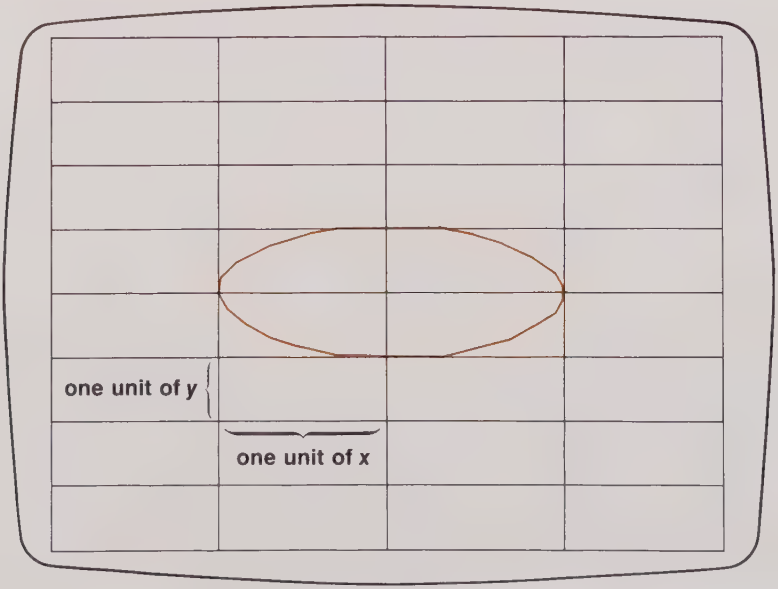
The scaling factors for X and Y are completely independent of each other. Therefore, plots are stretched or shrunk independently in the X and Y directions to fit the graphics display area *(anisotropic scaling)*. An X-unit of measure may not necessarily equal a Y-unit of measure.

Example: This program demonstrates the effects of the `SCALE` statement and unequal unit scaling on the plotting area. Note that the length of a unit-of-measure in the X and Y direction are not necessarily equal.

```
10 GCLEAR
20 DEG
30 SCALE -2,2,-4,4
32 ! DRAW A CIRCLE
40 MOVE 1,0
50 FOR A=0 TO 360 STEP 15
60 DRAW COS(A),SIN(A)
70 NEXT A
80 END
```

Clears the graphics display.
Sets degree mode.
Specifies X and Y units-of-measure.

Moves to start of circle.
`FOR-NEXT` loop to specify angle measures of circle.



Because of unequal unit scaling, our "circle" is shaped like an oval; one unit of X does not equal one unit of Y.

Equal Unit Scaling

Particularly with symmetrical plots and curves, it is important to scale the display proportionately in the X and Y directions so that one length of measure in the X direction will equal one length of measure in the Y direction *(isotropic scaling)*.

Since there are 256 useable dots in the horizontal direction and 192 useable dots in the vertical direction (a ratio of four X dots to three Y dots), scale the display so that the number of dots in a unit length of X is equal to the number of dots in a unit length of Y.

The actual ratio of intervals between dots on the display is 255 to 191. But, in most instances, you can use the following equation to determine the number of units in the X and Y directions for equal scaling:

$$X = {}^{4}/_{3}Y$$

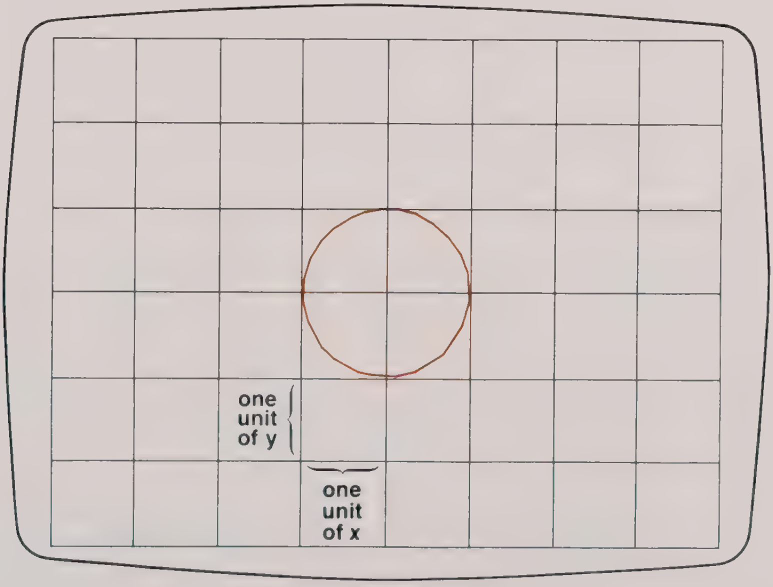
where: X is the number of units in the horizontal direction and
Y is the number of units in the vertical direction.

Example: Modify the `SCALE` statement from the last example so that the circle is drawn in correct proportions. One solution is to change statement 30 to read:

```
30 SCALE -4,4,-3,3
```

Scales 8 X-units by 6 Y-units; ratio of 4X to 3Y. Yields 32 dots per unit length of X and Y.

If you now run the modified program to generate a circle, the following will appear on the display:



As long as the number of dots per unit length of X is equal to the number of dots per unit length of Y, your plots will be drawn symmetrically in both X and Y directions.

More Examples of "Isotropic" Scaling:

`SCALE 0,4,0,3`	Scales 4 X-units by 3 Y-units; 64 dots per unit length.
`SCALE -6,10,-3,9`	Scales 16 X-units by 12 Y-units; 16 dots per unit length.
`SCALE -5,55,-5,40`	Scales 60 X-units by 45 Y-units; 4.262 dots per unit length.

Note: The exception to our rule of scaling $X = {}^4/_3Y$ is if you scale graphics display to the number of dots on the graphics screen. Then you should use the actual ratio of intervals, $X = {}^{255}/_{191}Y$.

`SCALE 0,255,0,191`	Scales 255 X-units by 191 Y-units (256 X dots by 192 Y dots).
or	
`SCALE 1,256,1,192`	

Both statements scale the graphics display so that one unit length is equal to the distance between two adjacent dots.

Drawing Coordinate Axes

The `XAXIS` and `YAXIS` statements draw an X-axis and a Y-axis, respectively, on the graphics display, with optional tic marks.

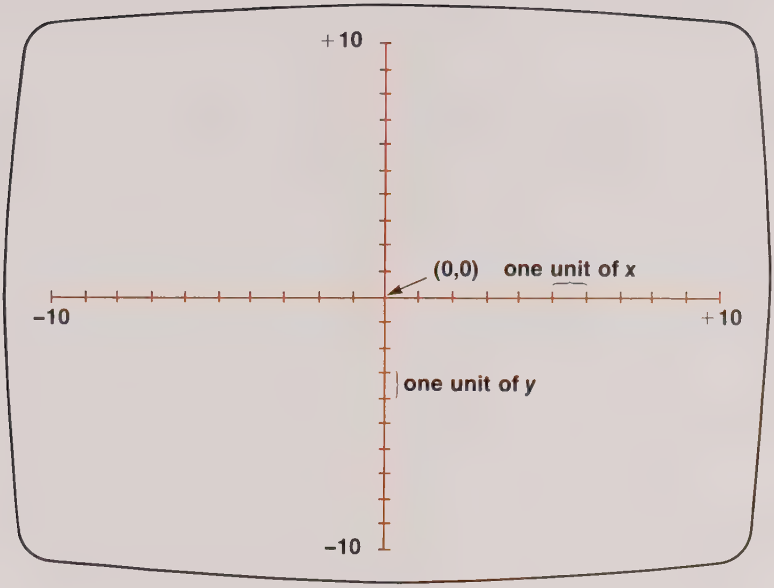
`XAXIS` Y-intercept[, tic spacing[, Xminimum , Xmaximum]]
`YAXIS` X-intercept[, tic spacing[, Yminimum , Ymaximum]]

The `XAXIS` statement generates an X-axis at the specified Y-intercept value on the display. The `YAXIS` statement generates a Y-axis at the specified X-intercept value on the display. An intercept value must be specified with an axis statement; the remaining parameters are optional.

The X and Y tic-spacing parameters are interpreted in the current scaled units. The sign of the tic-spacing parameter determines whether the tics will be drawn in increasing magnitude (positive) or decreasing magnitude (negative). For example, a negative tic parameter in an `XAXIS` statement means that tics will be drawn from right to left in the intervals specified.

Example: The following program first scales the display to be 20 X-units wide (from −10 to +10) and 20 Y-units long (from −10 to +10), then draws a pair of axes with tic marks at each scaled unit on the axes.

`10 GCLEAR`	Clears the graphics display.
`20 SCALE -10,10,-10,10`	Scales the graphics display.
● `30 XAXIS 0,1`	Draws an X-axis at Y-intercept 0, and marks one tic every X-unit.
● `40 YAXIS 0,1`	Then draws a Y-axis at X-intercept 0, and marks one tic every Y-unit.
`50 COPY`	
`60 END`	

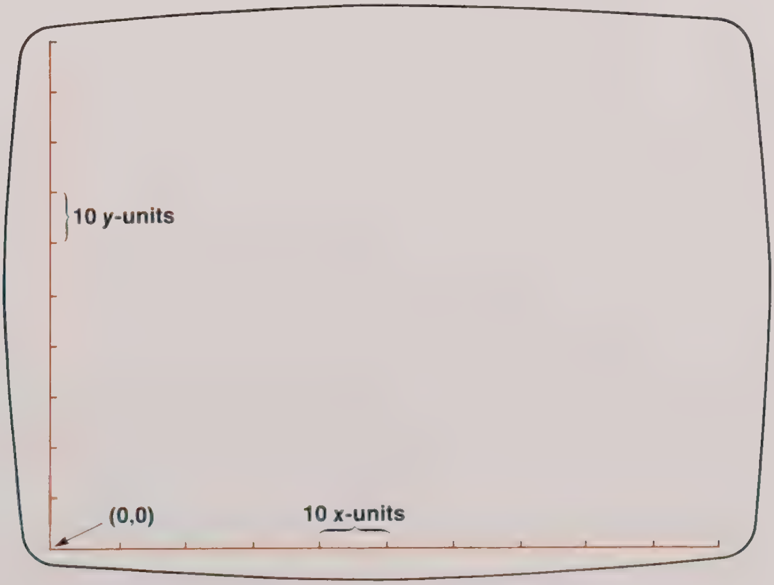

+10
(0,0)
one unit of x
-10
+10
one unit of y
-10

When the axes lie on the boundaries of the graphics display area, only half of each tic mark is shown; for example:

```
 10 GCLEAR
 20 SCALE 0,100,0,100

●30 XAXIS 0,10
●40 YAXIS 0,10
 50 COPY
 60 END
```

This is the default scale at power on or after RESET.

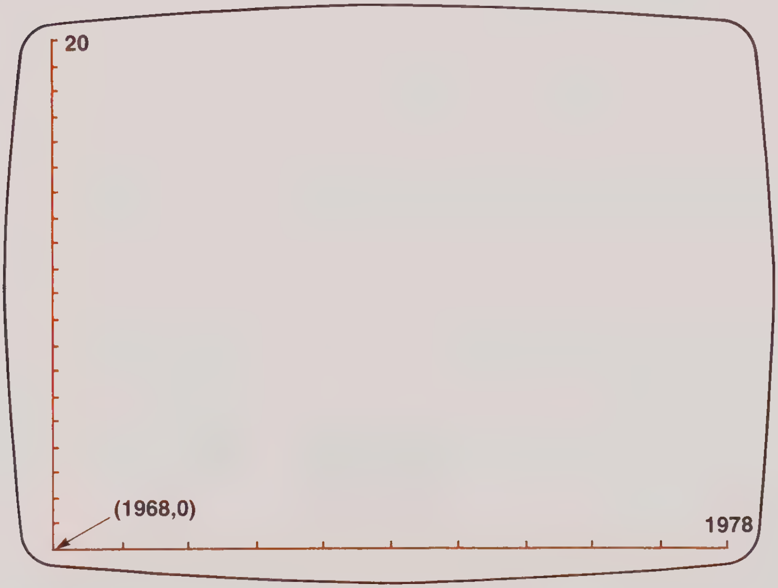
Draws an X-axis, marking tics every 10 units; draws a Y-axis, marking tics every 10 units, then copies the display onto paper.


10 y-units
(0,0)
10 x-units

To draw axes for the weather station graph, you might execute the following statements:

```
 10 GCLEAR
 20 SCALE 1968,1978,0,20
●30 XAXIS 0,1
●40 YAXIS 1968,1
 50 END
```

Notice that the origin has been scaled outside of the graphics display area.



A positive tic parameter instructs the system to draw tic-marks, at the specified interval, from left to right on the X-axis and from bottom to top on the Y-axis. In the example above, tics are drawn on the X-axis at 1968, 1969, 1970, …, 1978. On the Y-axis, tics are drawn at 0, 1, 2, …, 20.

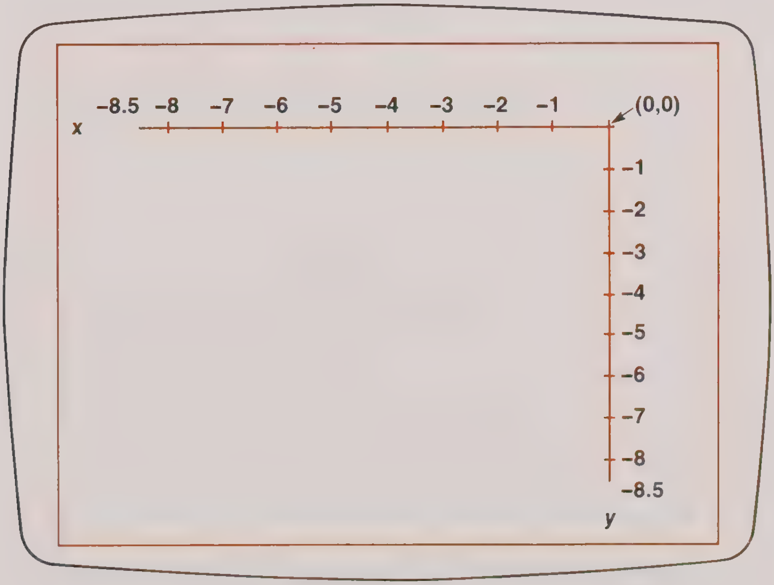
A negative tic parameter instructs the system to draw tics, at the specified interval, from right to left on the X-axis. If you use negative X- or Y-values, be aware of the sign of the tic parameters so that you space the tics correctly on the axes.

The minimum and maximum parameters specify the length of the axes within the current scale of the display. These parameters are especially useful when you want to allow space on the display for labels.

The following program illustrates the use of negative tic marks and maximum/minimum X-axis and Y-axis specifications:

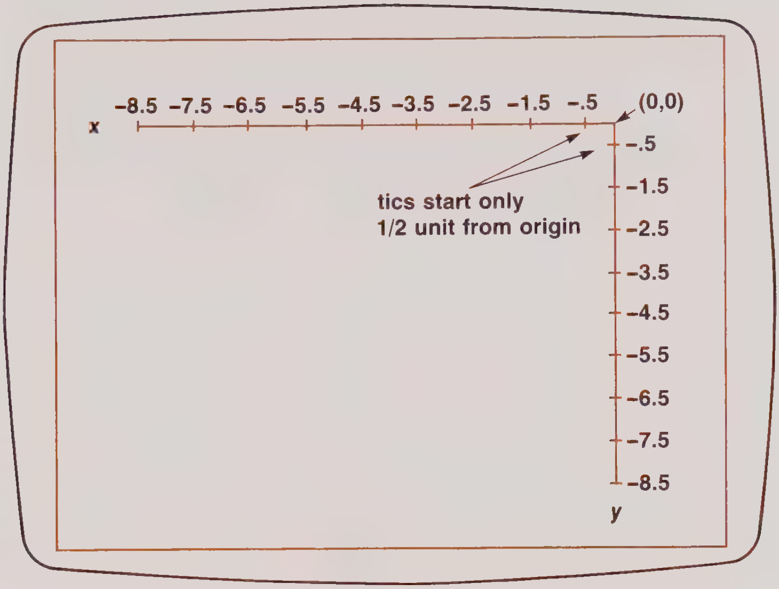
`10 GCLEAR` `20 SCALE -10,2,-10,2`	Scales the display at −10 to 2 from left to right, and −10 to 2 from bottom to top.
`●30 XAXIS 0,-1,-8.5,0`	Draws an X-axis at Y=0. Marks one tic for each X-unit from the right side of the axis to the left. Displays only that portion of the X-axis from −8.5 to 0.
`●40 YAXIS 0,-1,-8.5,0`	Draws a Y-axis at X=0. Marks one tic for each Y-unit from the top of the axis to the bottom. Displays the Y-axis from −8.5 to 0.

`50 XAXIS 2 @ XAXIS -10`
`60 YAXIS -10 @ YAXIS 2`
`70 END`
	Frames the display with a line at each of the graphics display boundaries.



If our program had been the following, using positive tic parameters, the tics would have been spaced incorrectly as shown:

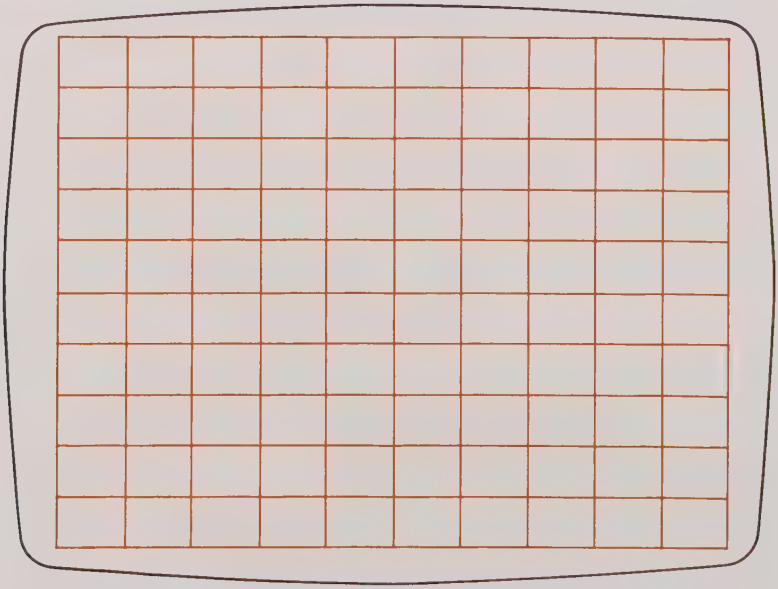
```
 10 GCLEAR
 20 SCALE -10,2,-10,2
●30 XAXIS 0,1,-8.5,0
●40 YAXIS 0,1,-8.5,0
 50 XAXIS 2 @ XAXIS -10
 60 YAXIS -10 @ YAXIS 2
 70 END
```


x
-8.5 -7.5 -6.5 -5.5 -4.5 -3.5 -2.5 -1.5 -.5
(0,0)
-.5
-1.5
-2.5
-3.5
-4.5
-5.5
-6.5
-7.5
-8.5
y
tics start only
1/2 unit from origin

As you have seen from our examples of framing the display, the axes statements may be used more than once in a program. In fact, the easiest way to draw a vertical or horizontal line is to use an axis statement specifying the X or Y position on the display.

For example, you might use the following program to draw and copy a grid of 10 X-units wide and 10 Y-units long.

```
10 GCLEAR
20 SCALE 0,10,0,10
30 FOR I=0 TO 10
40 XAXIS I @ YAXIS I
50 NEXT I
60 COPY
70 END
```



Plotting Operations

In the following pages, we present graphics statements that enable you to control the pen's movement and "color" in order to produce lines for graphic display.

PENUP

The `PENUP` statement lifts the pen so that you can move the pen without generating a line on the graphics display. Regardless of whether `PENUP` is executed from the keyboard or in a program, the statement's form is simply:

```
PENUP
```

Raises the pen; stops line generation.

The pen up or pen down status can be automatically controlled by using the `DRAW`, `MOVE`, `IDRAW`, and `IMOVE` statements.

PEN

The `PEN` statement specifies whether plotting is done with white dots or black dots. Thus, `PEN` enables you to draw lines and then erase them. The syntax for the `PEN` statement is:

`PEN` numeric expression

If the numeric expression is positive or zero, white dots are specified for plotting, black dots for clearing. If the numeric expression is negative, black dots are specified for plotting, white dots for clearing. The default pen status at power on or after pressing RESET is positive (white dots on black background) and `PENUP`.

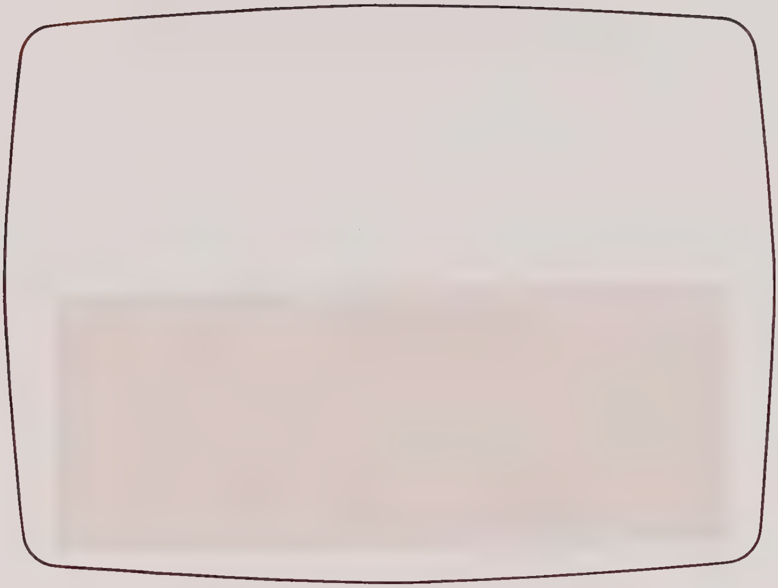
You can think of the pen as a drawing instrument with two colors of ink—black and white—and appropriate erasers for the background color. A positive pen number generates white lines on a black background. A negative pen number selects an "eraser" so that a line redrawn with a negative pen number will be erased with the color of the background. When a line is erased, the intersecting points of any intersecting lines will also be erased.

If you clear the graphics display following the execution of a negative pen number, that portion of the display specified by the `GCLEAR` statement will be cleared white.

For example, enter and execute the following program:

```
 10 SCALE 0,100,0,100
•20 PEN 1
 30 GCLEAR
•40 PEN -1

 50 GCLEAR 50
 60 END
```

Sets positive pen.
Clears the graphics display to black.
Specifies black plotting dots, white clearing dots.
Clears lower half of screen to white.



PLOT

The PLOT statement makes a dot at the specified X,Y coordinate position *or* draws a line to that position in current units using the current pen number.

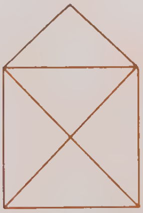
PLOT X-coordinate , Y-coordinate

The X and Y parameters are interpreted according to the current graphics display scale.

If the pen is up when PLOT is executed, the pen moves from the current point to the specified X,Y position, then drops to the screen, makes a dot, and stays down. If the pen is down when PLOT is executed, it stays down and draws a line from the current point to the specified point. If you do not wish to draw a line, the statement preceding PLOT should be a PENUP or MOVE statement.

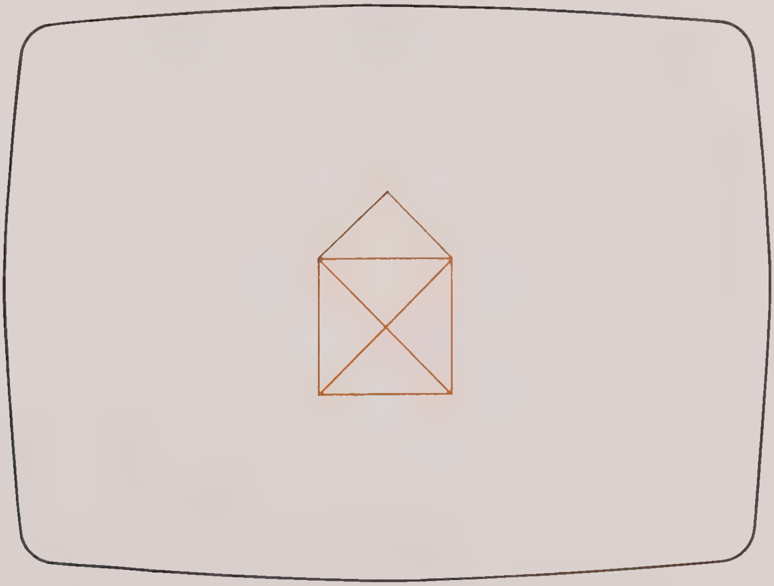
Example: Write a program to draw the figure below with these stipulations:

1. Once the pen is down, you cannot lift it until you have finished drawing the figure.
2. You cannot cross over any line that has been drawn previously.
3. No line can be drawn twice.



Your program might look like this:

`10 PEN 1 @ GCLEAR`	Clears graphics display.
`20 SCALE 0,20,0,15`	Equal unit scale; 20X by 15Y (ratio of 4X to 3Y).
`30 PENUP`	Lifts the pen.
`40 FOR I=1 TO 11`	Start of loop to plot figure.
`50 READ X,Y`	Reads coordinate values.
● `60 PLOT X,Y`	Plots accordingly.
`70 NEXT I`	Reads next values until done.
`80 DATA 8,5,8,9,10,11,12,9,8,9,10,7,12,9,12,5,10,7,8,5,12,5`	X,Y coordinate positions for `PLOT`.
`90 END`	

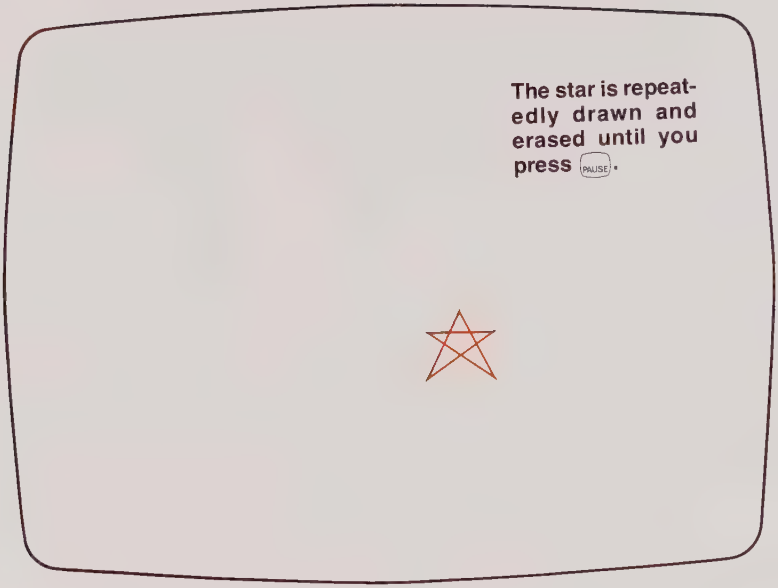


Example: Now write a program to generate a "twinkling" star on the display. First plot the star, then set up a loop to alternately erase it with the opposite pen color and plot it again.

Here's our solution:

```
  10 PEN 1 @ GCLEAR
  20 SCALE -10,10,-5,10
  30 P=1
  40 PENUP
  50 PEN P
  60 FOR I=1 TO 6
  70 READ X,Y
● 80 PLOT X,Y
  90 NEXT I
 100 P=-P
 110 RESTORE
 120 GOTO 50
 130 DATA 0,0,1,2,2,0,0,1,[illegible],2,1,4
     ,0,0
 140 END
```

Press PAUSE to stop the program.



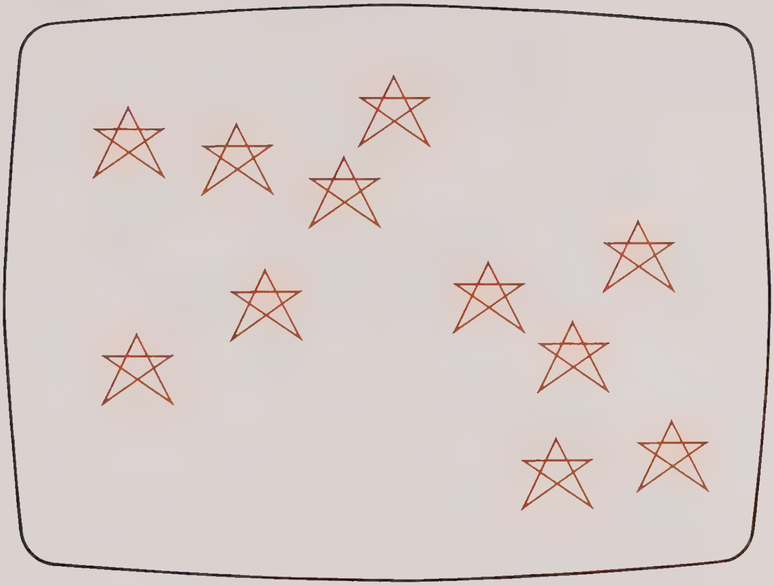
The star is repeatedly drawn and erased until you press PAUSE.

Example: Now that you have star drawing abilities, write a program to generate star clusters. **Hint:** Use the `RND` function to generate random increments for a given star pattern. In general, you can generate a sequence of random integers from a to b using the following formula: IP((b + 1−a)*RND+a).

Here's a sample solution:

```
  10 PEN 1 @ GCLEAR
  20 SCALE 0,20,0,15
  30 PENUP
  40 GOSUB 1000
  50 FOR I=1 TO 6
  60 READ X,Y
  70 PLOT X+X1,Y+Y1
  80 NEXT I
  90 RESTORE
 100 GOTO 30
 110 DATA 0,0,1,2,2,0,0,1.4,2,1.4
     ,0,0
1000 X1=18*RND
1010 Y1=13*RND
1020 RETURN
1030 END
```

Each time you run the program, you can generate a different "constellation." Press PAUSE to stop the program, CONT to continue with the current display, and RUN to begin with a clear screen.



Moving and Drawing

The most useful of the graphics statements, `DRAW` and `MOVE`, automatically control the pen up or down positions. We will discuss the most efficient way to use them on page 212.

MOVE

The `MOVE` statement lifts the pen and then moves the pen to the specified X,Y coordinate position in current units and leaves the pen up. This statement provides an easy way of moving the pen without drawing a line on the graphics display, regardless of whether the pen is currently up or down.

`MOVE` X-coordinate , Y-coordinate

The X and Y parameters are interpreted according to the current scaled units.

Example program lines:

```
30 MOVE 2,5
60 MOVE 25,50
```

Moves with pen up to point 2,5.
Moves with pen up to 25,50.

DRAW

The `DRAW` statement drops the pen and then draws a line to the specified X,Y coordinate position in current units. This statement provides an easy way of drawing a line from the current pen's location to a new location regardless of whether the pen is currently up or down.

`DRAW` X-coordinate , Y-coordinate

The X and Y parameters are interpreted according to the current scaled units.

Example program lines:

`20 DRAW 2,5`	Draws a line from current pen location to point 2,5.
`70 DRAW 25,50`	Draws a line from current pen location to point 25,50.

Drawing Curves

The concept of incremental drawing proves extremely useful when implemented to draw curved figures. As you know, you can approximate curves with line segments; many short line segments approximate a curve better than several long line segments.

With HP-85 graphics, you always plot directly from one X,Y coordinate position to another, in this way generating ``lines'' which are actually a series of dots in a straight line. Since you do not have a pen with ink to draw a continuous curve, you must evaluate the equation for a curve in small enough intervals to generate enough ‘‘line segments’’ to simulate the curved figure.

This can be done very easily in BASIC programming language with a `FOR-NEXT` loop using a `STEP` interval. How small of an interval is small enough? This, of course varies with the curve you wish to display. Generally, 20 to 30 intervals provide enough points to adequately plot a curve.

Earlier in this section, we plotted a circle using a `FOR-NEXT` loop using a `STEP` interval. Below, we have rewritten the program for a circle to illustrate our discussion. The first loop computes the step value; in other words, it determines the number of points that will be used to plot the circle. As the step value becomes smaller, the figure displayed makes a closer approximation to a circle.

```
 10 DEG
 20 PEN 1 @ GCLEAR
 30 SCALE -4/3,4/3,-1,1
 40 ! SET INCREMENT VALUE
 50 FOR I=4 TO 30 STEP 2
 60 S=360/I
 70 ! NOW DRAW A CIRCLE
 80 MOVE 1,0
 90 FOR A=0 TO 360 STEP S
100 DRAW COS(A),SIN(A)
110 NEXT A
120 MOVE 0,0
130 LABEL "I="&VAL$(I)
140 WAIT 3000
150 PEN -1
160 MOVE 0,0
170 LABEL "I="&VAL$(I)
180 PEN 1
190 NEXT I
200 END
```

Scale changed to draw a larger circle.

Sets the step increment value.

Draws the circle; plots as many points as `STEP` will allow.

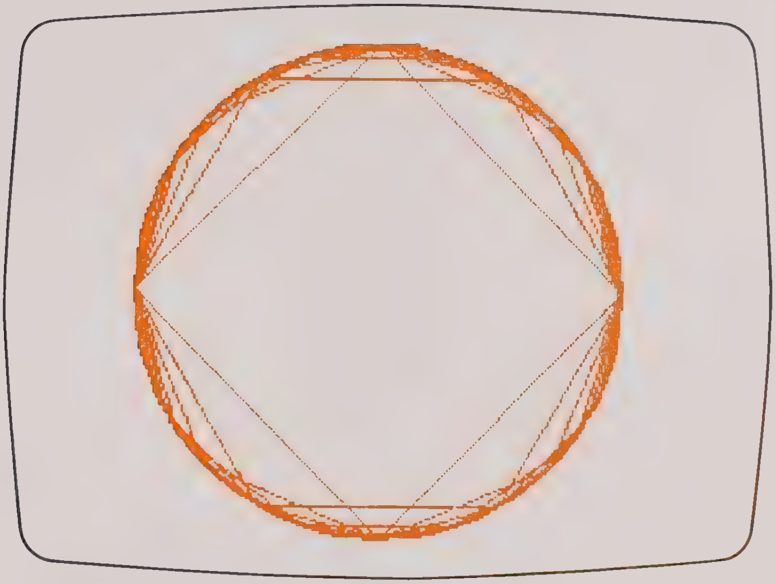
We’ll discuss this statement in the next section.

Displays circle for approximately 3 seconds.

Now erases the label by selecting the opposite pen color and relabeling.

Enter the program and press (RUN) to execute it. If you wish to have a printed copy of the figure on the display, just press the (COPY) key to copy the display. Remember you can press (COPY) while a program is running without interrupting program execution.

Below is the plot for I values from 4 to 30 in increments of two.



As you ran the program, you may have noticed that there were very small differences in the circle between values of I from 24 to 30. As you become more familiar with graphing curves on the HP-85 you'll become a better judge of the number of intervals that are necessary to plot a curve.

If you choose an increment value that is too small, it may take the system a long time to plot the graph or curve. You can stop program execution at any time by pressing (RESET), and then edit your increment values if you wish.

Padding the Increment Loop

At this point, we digress a moment from our discussion of the graphics statements to point out an important concept about drawing lines with FOR-NEXT loops. If you thoroughly understand the STEP incrementing process with loops, skip to the problems on page 216.

When a fractional number of increment intervals are specified to complete a graphics figure, it is often necessary to "pad" the final value of the loop counter so that the figure is drawn completely.

Example: The equation for a cardioid is:

$$r = a(1 - \cos\theta)$$

where;

r is the directed distance from the origin to a point on the curve,
a is any positive constant, and
θ is the angle measure.

Write a program that plots a cardioid of the form $r = 1 - \cos\theta$ in radians mode and copies it on the printer.

Suppose your program looked like this:

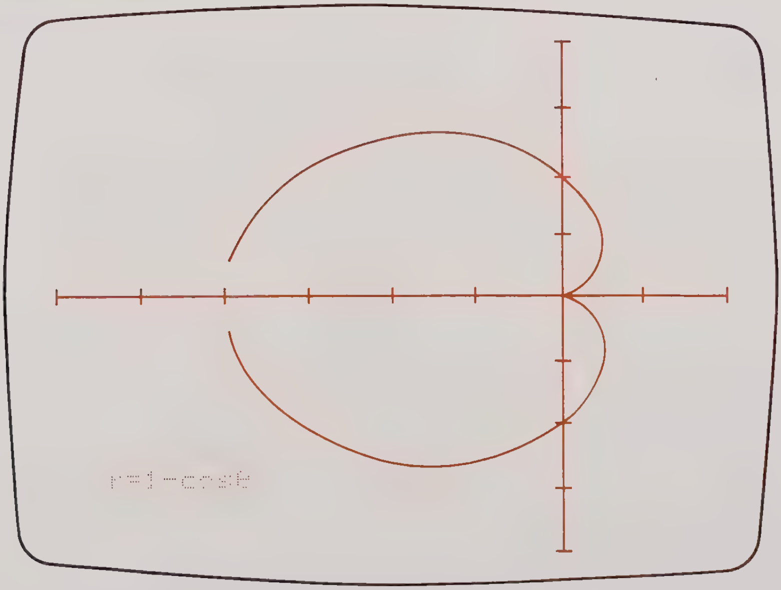
```
  10 PEN 1 @ GCLEAR
  20 SCALE -3,1,-2,2
  30 XAXIS 0,.5
  40 YAXIS 0,.5
  50 RAD
  60 MOVE 0,0
● 70 FOR T=0 TO PI STEP .15

  80 R=1-COS(T)
  90 DRAW R*COS(T),R*SIN(T)
 100 NEXT T
 110 MOVE 0,0
●120 FOR T=0 TO -PI STEP -.15
 130 R=1-COS(T)
 140 DRAW R*COS(T),R*SIN(T)
 150 NEXT T
 160 MOVE -2.75,-1.5
*170 LABEL "r=1-cosθ"
 180 END
```

Isotropic scale.
Tic marks every 1/2 unit.

Sets radians.
Moves to point 0,0.
Begins plotting cardioid in increments of 0.15 radians.
Polar/rectangular coordinate conversions.

Move to point 0,0 to plot the other half of the cardioid.

Move to point −2.75, −1.5.
Label graph. (Again, we'll discuss labeling in the next section).



---

* Type character θ by pressing CTRL P.

As you can see, the figure was not completely drawn. Parts of the curve closest to the value of π were omitted. Examine the `FOR-NEXT` loops:

```
 70 FOR T=0 TO PI STEP .15
.
.
.
100 NEXT T
120 FOR T=0 TO -PI STEP -.15
.
.
.
150 NEXT T
```

This loop is executed at T=0, T=0.15, T=0.3, ..., continuing in increments of 0.15 through T=3. But when T=3+0.15 =3.15, which is greater than the final value of the loop counter (π), the program exits the loop. The fractional part of π is not evaluated.

Similarly, the second loop does not evaluate the portion of the curve closest to −π below the X-axis.

In both loops, when the absolute value of T is greater than π, the program exits the loop.

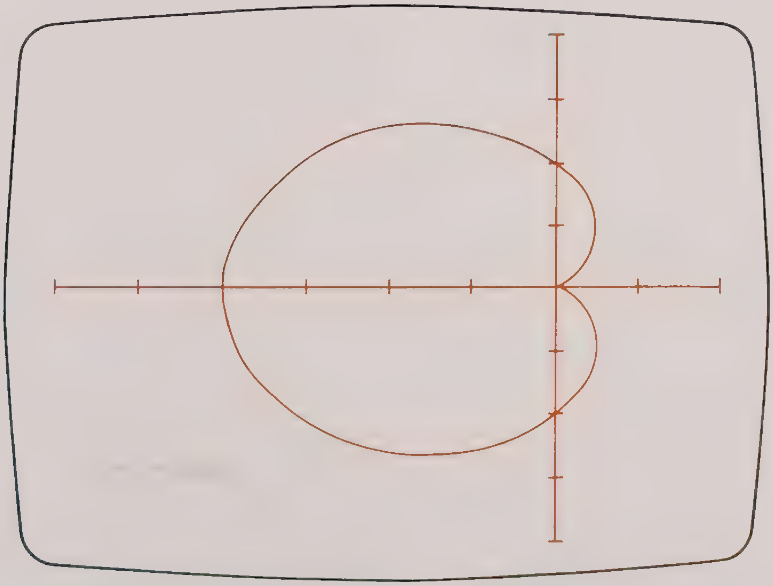
You can correct the effect of the increment value by "padding" the final value of the loop counter. In the cardioid program, extend the final value that follows `TO` in statements 70 and 110 by 0.1, so that they read:

```
 70 FOR T=0 TO PI+.1 STEP .15
:
120 FOR T=0 TO -PI-.1 STEP -.15
```

Add 0.1 to π here.

Subtract 0.1 from −π here.

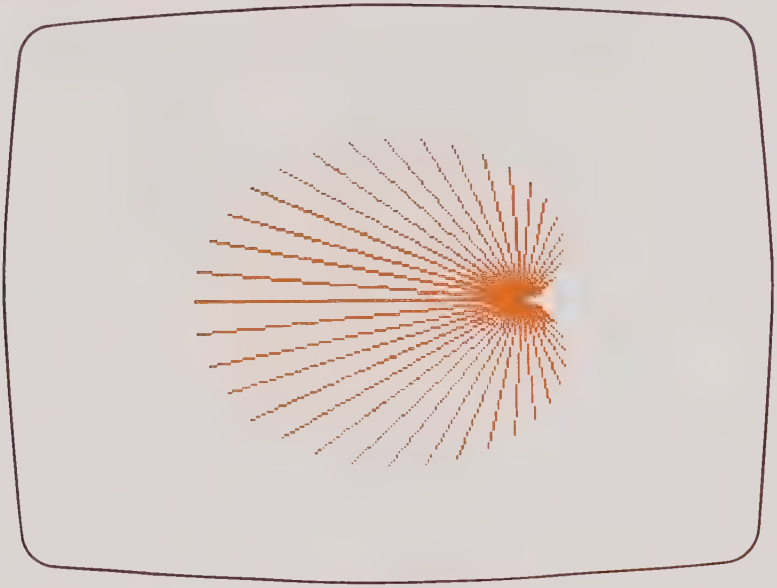
Now the full range of values for each loop will be evaluated. Run the program again to display and copy the completed cardioid.



The first loop is executed at T=3.15 because T is still less than π + 0.1; the second loop is executed at T=−3.15 because T is greater than −π −0.1.

Problems

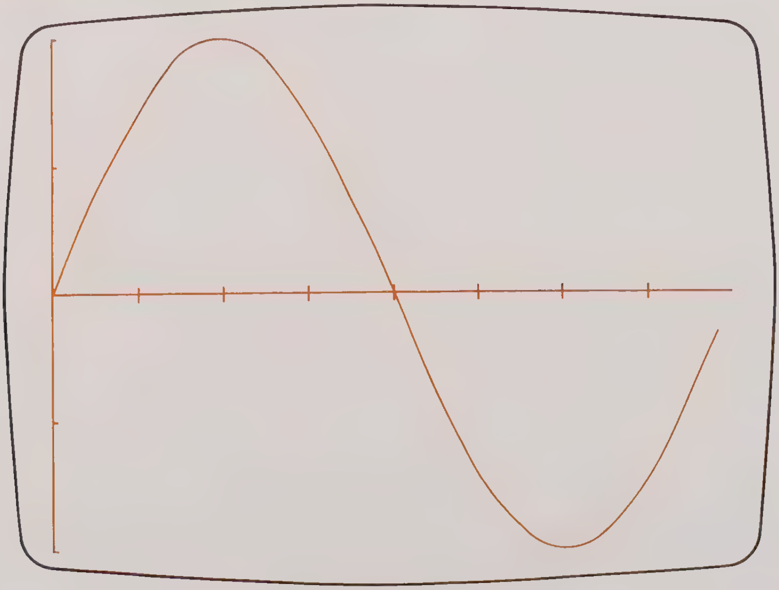
12.1 Now that you've had some experience in graphing cardioids, write a program to display the following:



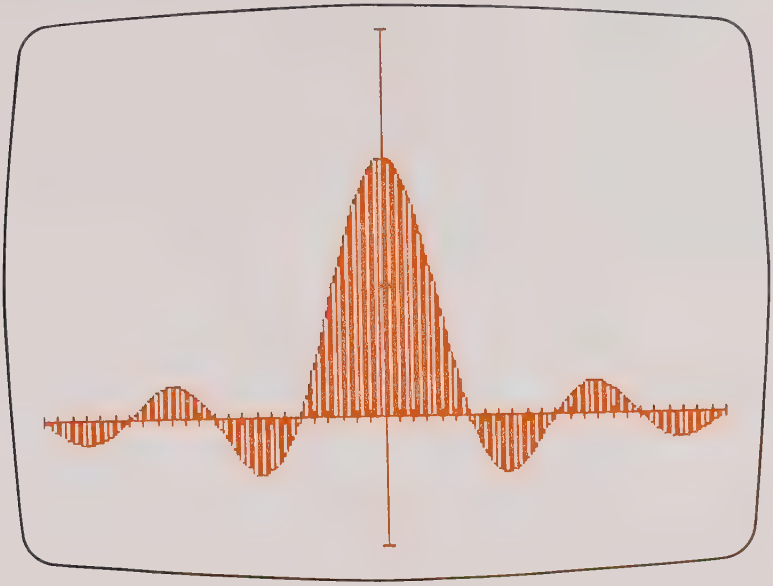
12.2 Pad the `FOR-NEXT` loop in the following program to complete the sine curve.

```
 10 PEN 1 @ GCLEAR
 20 SCALE 0,2*PI,-1,1
 30 XAXIS 0,PI/4
 40 YAXIS 0,.5
 50 RAD
 60 MOVE 0,0
 70 FOR X=0 TO 2*PI STEP PI/20
 80 DRAW X,SIN(X)
 90 NEXT X
100 END
```

Moves to start of curve.



12.3 Write a program to generate the curve of the SIN (X)/X from -4π to $+4\pi$. As you plot the curve, also draw "fill" lines from the curve to the X-axis. Be sure to check for X=0, so that you don't divide by zero.



IMOVE

The `IMOVE` *(incremental move)* statement provides incremental moving capability. The origin is assumed to be the current pen position.

`IMOVE` X-increment , Y-increment

The `IMOVE` statement interprets the X and Y parameters according to the current scaled units relative to a local origin. The local origin is that of the pen position before the `IMOVE` statement is executed (i.e., the current pen position).

Thus, the `IMOVE` statement moves the pen, without drawing a line, from the current pen position to that position plus or minus the increment in each coordinate value.

Example program statements:

`50 IMOVE 1,3`	Moves the pen from current pen position (say X,Y), one unit to the right and three units up (or, to point X + 1, Y + 3).
`80 IMOVE -5,2`	Moves the pen from current pen position (X,Y), five units to the left and two units up (to point X−5, Y+2).

IDRAW

The `IDRAW` *(incremental draw)* statement provides incremental drawing capability. The origin is assumed to be the current pen position.

`IDRAW` X-increment , Y-increment

The `IDRAW` statement interprets the X and Y parameters according to the current scaled units relative to a local origin. The local origin is that of the pen position before the `IDRAW` statement is executed (i.e., the current pen position).

Thus, the `IDRAW` statement draws a line from the current pen position to that position plus or minus the increment in each coordinate value.

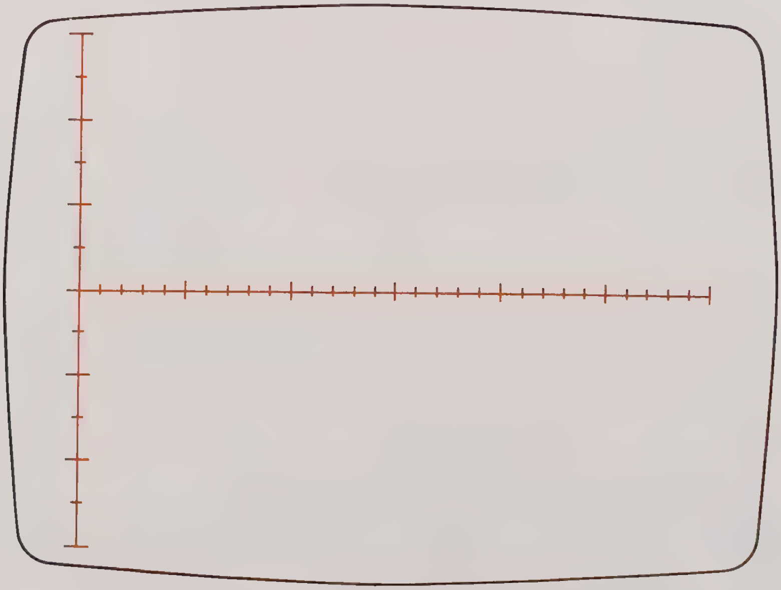
Example program statements:

`30 IDRAW 1,3`	Draws a line to a point one unit to the right and three units up from current pen position.
`60 IDRAW -5,2`	Draws a line to a point five units to the left and two units up from current pen position.

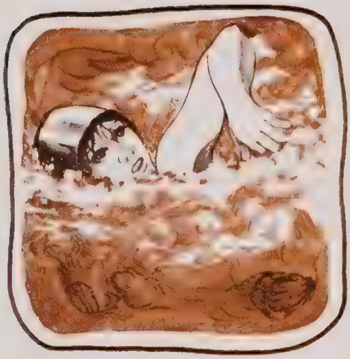
The `IMOVE` and `IDRAW` statement are particularly useful for plotting lines or figures of similar slope and size when the exact coordinate positions are unknown.

For instance, suppose you wish to make larger tic marks on the X-axis, every five units, and on the Y-axis, every two units. You might write a program like this.

```
  10 PEN 1 @ GCLEAR
  20 SCALE -2,30,-6,6
  30 XAXIS 0,1,0,30
  40 YAXIS 0,1
  50 FOR X=0 TO 30 STEP 5
  60 MOVE X,.2
● 70 IDRAW 0,-.4
  80 NEXT X
  90 FOR Y=-6 TO 6 STEP 2
 100 MOVE -.5,Y
●110 IDRAW 1,0
 120 NEXT Y
 130 END
```



Example: Using the information from the table below, graph the Summer Olympic records for the 100-meter freestyle swimming—men and women—from 1948 to 1976. Instead of plotting a point, make a "+" symbol for each of the women's records and "□" symbol for each of the men's records. (Since we will use this example later in the section, you may wish to store the program on tape after you've entered it into computer memory.)



Summer Olympics Winners and Records: 100-Meter Freestyle

Year	Men	Time (seconds)	Women	Time (seconds)
1948	Ris	57.3	Anderson	66.3
1952	Scholes	57.4	Szoke	66.8
1956	Henricks	55.4	Frazer	62.0
1960	Devitt	55.2	Frazer	61.2
1964	Schollander	53.4	Frazer	59.5
1968	Wenden	52.2	Henne	60.0
1972	Spitz	51.22	Neilson	58.59
1976	Montgomery	49.99	Ender	55.65

```
  10 REM *100-METER FREESTYLE SUM
     MER OLYMPICS 1948-1976
  20 GCLEAR
  30 SCALE 1940,1978,42,70.5
  40 XAXIS 48,4,1944,1978

  50 YAXIS 1944,1,48,70
  60 FOR I=1948 TO 1976 STEP 4
  70 READ M

  80 GOSUB 1000
  90 READ W

 100 GOSUB 2000
 110 NEXT I
 120 DATA 57.3,66.3,57.4,66.8,55.
     4,62,55.2,61.2,53.4,59.5,52.
     2,60,51.22,58.59,49.99,55.65
 130 STOP
 1000 REM *PLOT SQUARE*
 1010 MOVE I,M
•1020 IMOVE -.2,.2
•1030 IDRAW .4,0 @ IDRAW 0,-.4
•1040 IDRAW -.4,0 @ IDRAW 0,.4
 1050 RETURN
 2000 REM *PLOT CROSS*
 2010 MOVE I,W
•2020 IMOVE 0,.3 @ IDRAW 0,-.6
•2030 IMOVE .3,.3 @ IDRAW -.6,0
 2040 RETURN
 3000 END
```

Equal unit scaling. (30)

Displays X-axis from 1944 to 1978 with tics every 4 years. (40)

Displays Y-axis from 48 to 70 with tics every second. (50)

Reads men's record in year set by loop counter. (70)

Go plot square. (80)

Reads women's record in year set by loop counter. (90)

Go plot cross. (100)

Moves to year, men's record in seconds. (1010)

Moves to a point −0.2 left and 0.2 up. (1020)

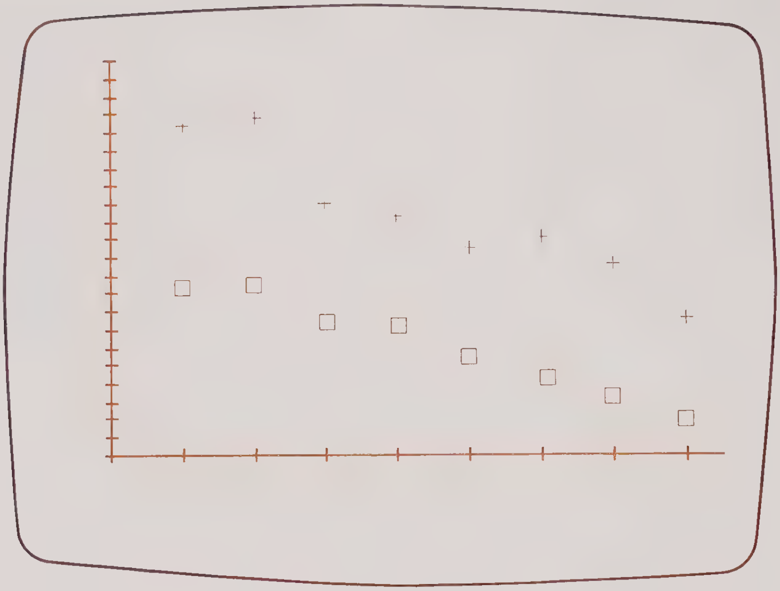
Draws top and right side of square. (1030)

Draws bottom and left side of square. (1040)

Moves to year, women's record in seconds. (2010)

Moves 0.3 unit up then draws line down. (2020)

Moves to a point 0.3 unit up and 0.3 unit right, then draws line left. (2030)



Problem

12.4 The following program, using `IDRAW`, generates some interesting graphic designs based on the form of a hyperbolic spiral. What angle increments generate the most interesting designs? Why do we include statement 120? How would you modify the program to generate a spiral twice as wide?

```
 10 DEG
 20 CLEAR
 30 DISP "INVERSE VIDEO: YES OR
    NO";
 40 INPUT P$
 50 IF P$="YES" THEN PEN -1 ELSE
    PEN 1
 60 SCALE -36000,36000,-36000,36
    000
 70 DISP "ANGLE INCREMENT VALUE"
    ;
 80 INPUT A
 90 GCLEAR
100 MOVE 0,0
110 FOR I=0 TO 36000 STEP A
120 IF A>90 THEN J=2*I ELSE J=I
130 IDRAW J*COS(I),J*SIN(I)
140 NEXT I
150 END
```

RUN

```
INVERSE VIDEO: YES OR NO?
YES
ANGLE INCREMENT VALUE?
91
```

Labeling Graphs

As you have seen from the "circle approximation" program and the "cardioid" program, you can further enhance the legibility of data plots by labeling graphs using the `LABEL` statement.

`LABEL` string expression

Note that only string expressions may be used with the `LABEL` statement. The string expression may include quoted character strings, string variables, string functions, substrings, and the string concatenator, `&`. The size of the character(s) specified with the `LABEL` statement is the same on the graphics display as the alpha display. Examples of `LABEL` statements that we have used already are:

```
  30 SCALE -4/3,4/3,-1,1
 .
 .
 .
 120 MOVE 0,0
• 130 LABEL "I="&VAL$(I)
```

Example from page 212.

Moves to point 0,0.
Then labels.

```
  20 SCALE -3,1,-2,2
 .
 .
 .
 160 MOVE -2.75,-1 5
• 170 LABEL "r=1-cosθ"
```

Example from page 214.

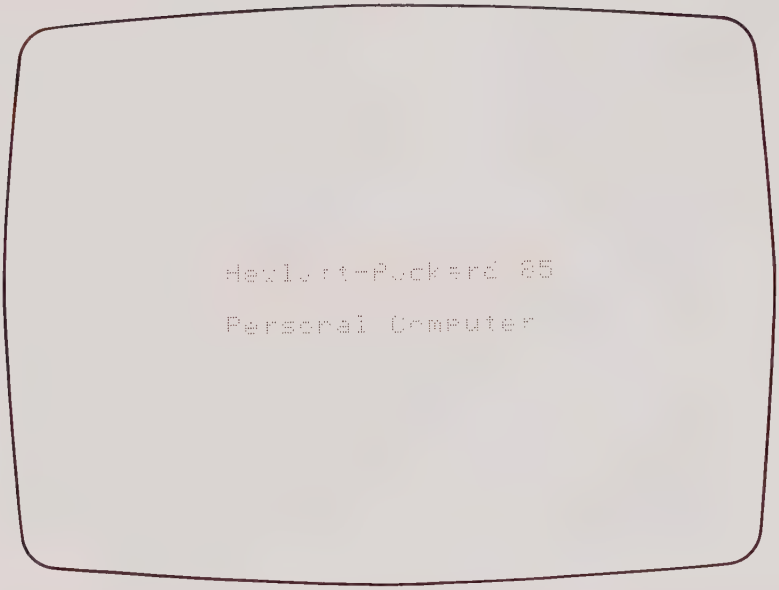
Moves to point −2.75, −1.5.
Then labels.

In each of the examples, we first direct the pen to the starting position of the label, then we specify the expression.

Example: Enter and run the following program.

```
  10 GCLEAR
  20 SCALE -1,1,-1,1
  30 MOVE -.5,.1
• 40 LABEL "Hewlett-Packard 85"
  50 MOVE -.5,-.1
• 60 LABEL "Personal Computer"
  70 END
```

Clears graphics display.
Scales X and Y units.
Moves to point −0.5,0.1.
Writes expression on graphics display.
Moves to point −0.5,−0.1.
Writes expression.



Hewlett-Packard 85

Personal Computer

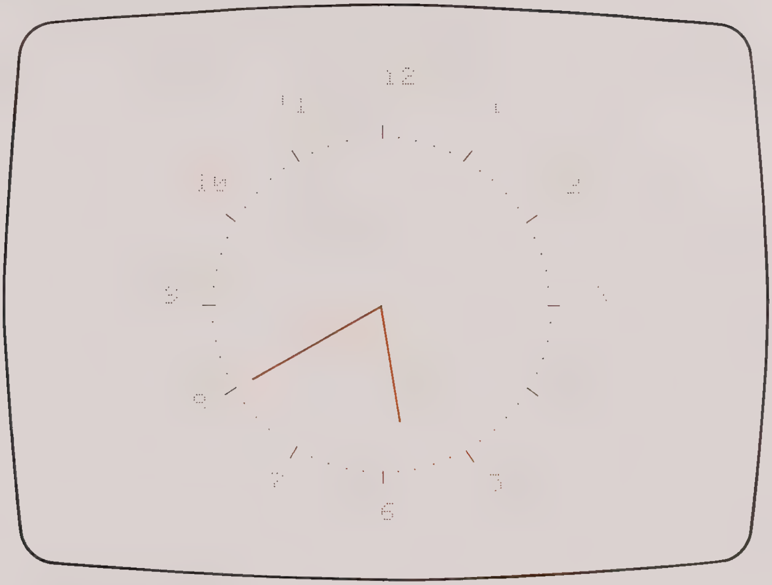
Example: Draw and label the face of a clock. Include a subroutine to draw the hour hand and the second hand for a time that you input. Write the program in such a way that the old time will be erased before a new time is drawn.

```
  10 PEN 1 @ GCLEAR                      Specifies positive pen; clears graphics
                                         display.
  20 SCALE -2,2,-3/2,3/2                 Equal unit scale.
  30 DEG                                 Sets degrees mode.
  40 ! FACE OF CLOCK
  50 FOR M=0 TO 360 STEP 6               Draws minute marks.
  60 MOVE SIN(M),COS(M)
  70 IDRAW SIN(M)/50,COS(M)/50
  80 IF M MOD 5 THEN 100                 Draws larger marks for 5-minute intervals.
  90 IDRAW SIN(M)/15,COS(M)/15
 100 NEXT M
 110 FOR I=1 TO 12
 120 MOVE 1.3*SIN(30*I),1.3*COS(3        Labels the clock with hours.
     0*I)
 130 LABEL VAL$(I)
 140 NEXT I
 150 ALPHA                               Puts display in alpha mode for input.
 160 DISP "INPUT TIME; HH.MM"
 170 INPUT T                             Inputs time in form HH.MM.
 180 GOSUB 1000                          Goes to subroutine to draw hands.
 190 PAUSE                               Pauses to display clock.
 200 PEN -1                              Specifies negative pen.
 210 GOSUB 1000                          Gosub to erase hands.
 220 PEN 1                               Specifies positive pen again.
 230 GOTO 150                            Goes back to line 150 to input new time.
1000 REM *DRAW HANDS OF CLOCK*
1010 MOVE 0,0                            Moves to middle of clock.
1020 H=30*IP(T)
1030 M=6*FP(T)*100
1040 DRAW .7*SIN(H+M/12),.7*COS(         Shorter hour hand.
     H+M/12)
1050 MOVE 0,0
1060 DRAW .9*SIN(M),.9*COS(M)            Larger minute hand.
1070 RETURN
```

Run the program for times of 5:40 and 10:15:

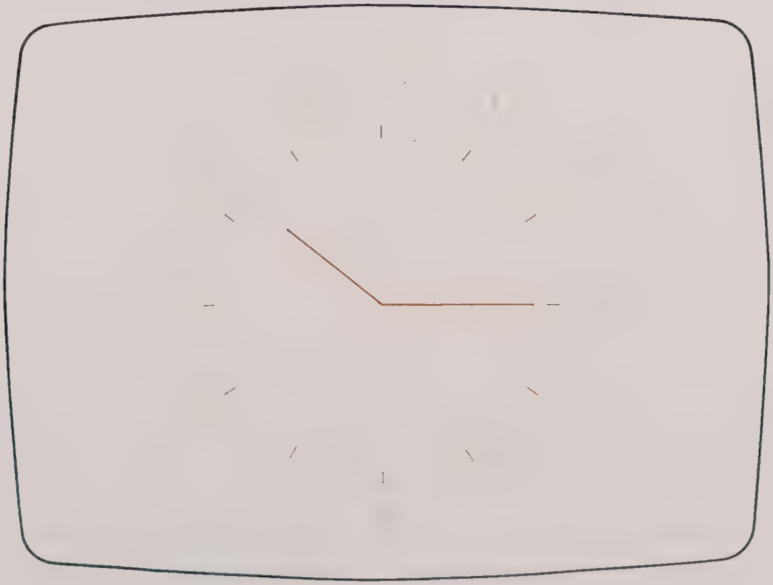
RUN

```
INPUT TIME; HH.MM
?
5.40
```


12
11
1
10
2
9
8
7
5
6

CONT

```
INPUT TIME; HH.MM
?
10.15
```



Label Direction

Character positions are much more flexible in graphics mode than alpha mode. Labels can be positioned either vertically or horizontally by using the `LDIR` *(label direction)* statement.

`LDIR` numeric expression

If the expression has a rounded integer value less than 45, labels will be positioned horizontally. If the value of the numeric expression (rounded to an integer) is greater than or equal to 45, labels will be positioned vertically. Thus:

`LDIR 0`	Specifies horizontal labels.
`LDIR 90`	Specifies vertical labels.

Example:

```
 10 SCALE -10,10,-10,10
 20 ALPHA
 30 DISP "ENTER A NUMBER FROM 0
    THROUGH 90"
 40 INPUT D
 50 PEN 1 @ GCLEAR
 60 LDIR D
 70 MOVE 0,0
 80 LABEL "---LDIR"&VAL$(D)
 90 PAUSE
100 GOTO 20
110 END
```

Line 20: Sets display to alpha mode for displaying input message.

Line 40: Inputs label direction.

Line 50: Clears graphics display.

Line 60: Sets label direction.

Line 70: Moves pen to point 0,0.

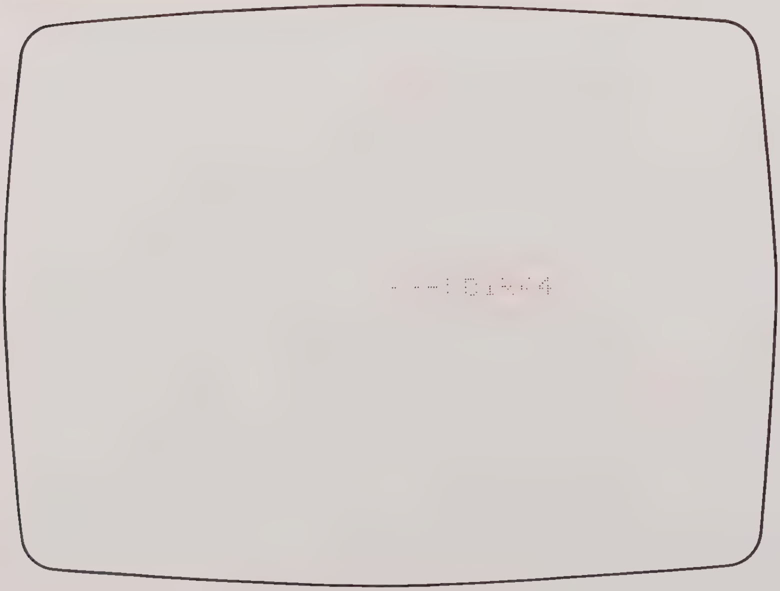
Line 80: Writes label on graphics display.

Run the program above with test values of D. Press CONT each time you wish to enter a new label direction.

We run the program with values of 44 and 45.

RUN

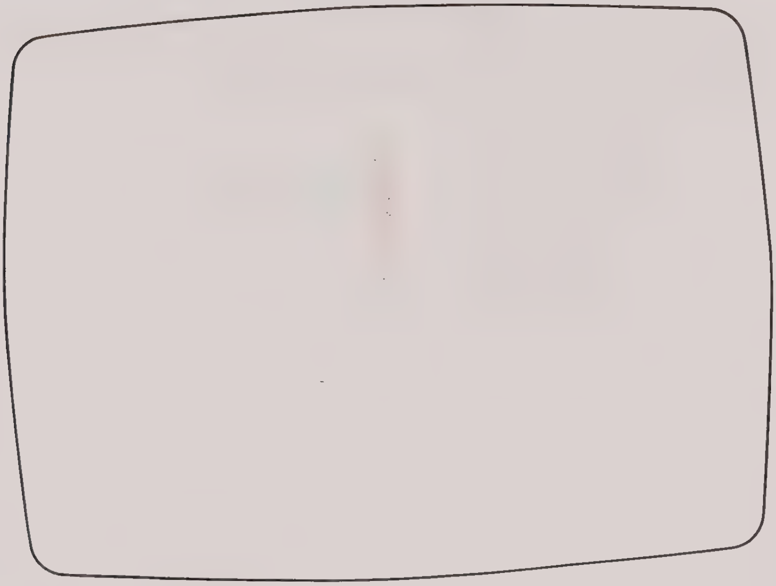
```
ENTER A NUMBER FROM 0 THROUGH 90
?
44
```



`LDIR44` yields a horizontal label.

CONT

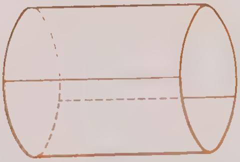
```
ENTER A NUMBER FROM 0 THROUGH 90
?
45
```



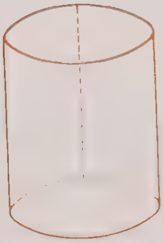
`LDIR45` yields a vertical label.

Label Length

You can think of the alpha display as a cylinder with four displays connected from top to bottom. But the graphics display treats characters as if it were a cylinder composed of *one* display with the right and left edges connected:



Alpha Display



Graphics Display

Thus, characters or lines do not cause the graphics display to scroll. In fact, characters will be chopped off if they are positioned too high on the graphics display. If vertical labels are positioned too far to the left of the display, part of the label will be written on the right boundary of the graphics display. A horizontal label longer than 32 characters will wrap around on top of itself, one dot below the original starting position.

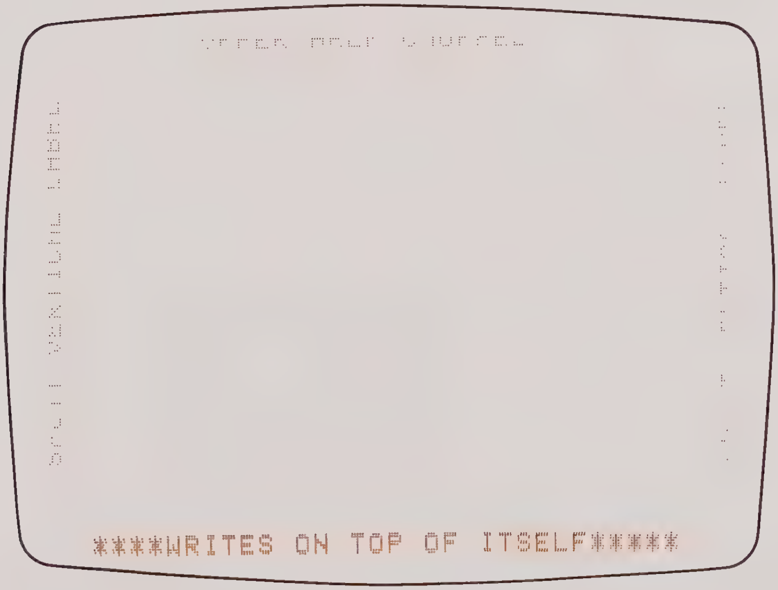
Example: This program illustrates the effects of the graphics display on character labels.

```
  10 GCLEAR
  20 SCALE 0,10,0,10
  30 LDIR 0
  40 MOVE 1,9.8
● 50 LABEL "    UPPER HALF CHOPPED
  "
  60 MOVE 0,0
● 70 LABEL "***WRITES ON TOP OF
    ITSELF*********WRITES ON TOP
     OF ITSELF*****"
  80 LDIR 90
  90 MOVE .1,1.2
●100 LABEL "SPLIT VERTICAL LABEL"
 110 END
```

Horizontal label setting.
Moves to point 1,9.8.
Writes lower part of label.

Moves to point 0,0.
Writes label.

Changes label direction.
Moves to point 0.1,1.2.
Writes label.



A label direction setting will remain the same until it is changed by another LDIR statement. Unless otherwise specified, labels will automatically be positioned horizontally. As we shall see, any input in graphics mode resets LDIR to horizontal labeling.

If a vertical label begins at the bottom of the display, a maximum of 24 characters will be written on the graphics display. Any remaining characters will be chopped off of the top of the display.

Positioning Labels

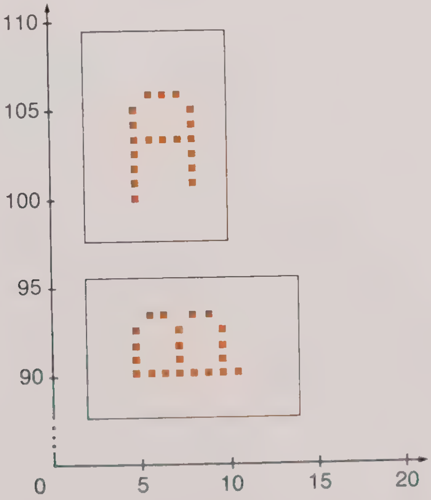
The position of a label is determined both by the `LDIR` statement and by the current pen location. Horizontal labels begin directly above the point specified by the current pen location. Vertical labels begin directly to the left of the specified pen location.

It is often easier to scale the display to the number of plotting dots available to specify exact label locations, from 0 to 255 in the horizontal direction (256 dots) and from 0 to 191 in the vertical direction (192). Since a character is composed of a 5 × 7 dot character on an 8 × 12 dot field, you can easily calculate the number of dots necessary for a particular label.

To illustrate label starting positions, we ran the following program and then enlarged the display area around the labels.

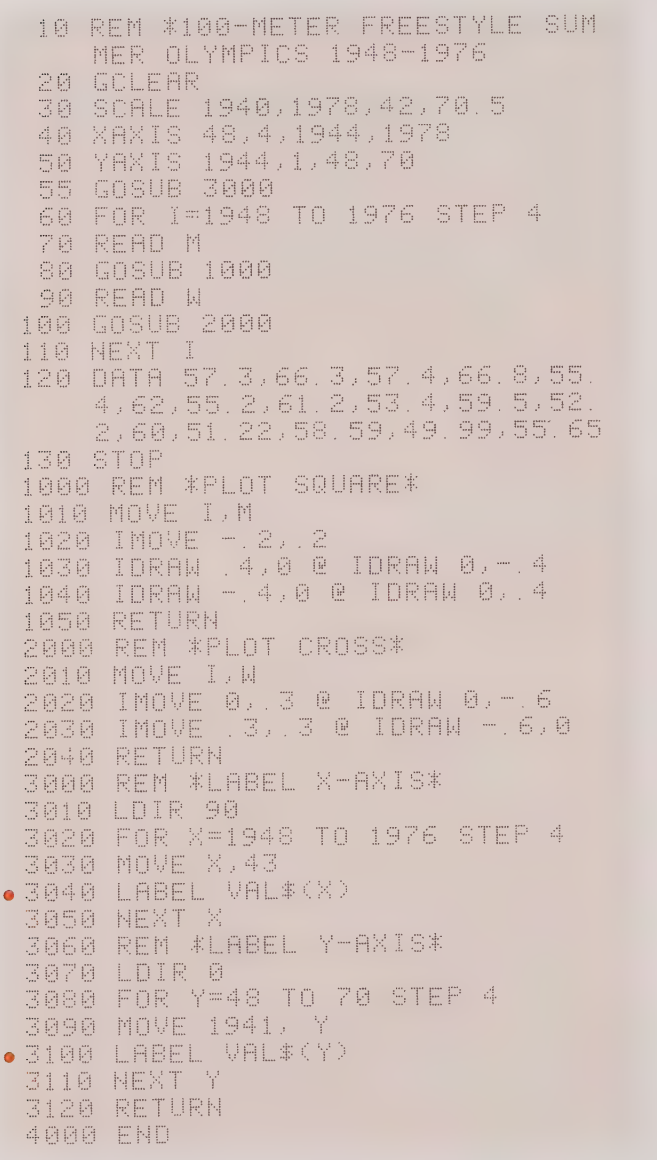
```
 10 GCLEAR
 20 SCALE 0,255,0,191
 30 PENUP
 40 PLOT 4,100
 50 LDIR 0
 60 LABEL "A"
 70 PENUP
 80 PLOT 11,90
 90 LDIR 90
100 LABEL "B"
110 END
```

Let's look at the section of the display around the labels:



A label is positioned directly above (horizontal) or to the left (vertical) of the current pen location to allow for underscored characters (character codes 128 through 255). Since we plotted point (4,100) immediately before the `LABEL` statement, `A` was positioned as shown above. If we had plotted or moved to (25,90) for instance the `A` would be positioned so that the left leg of the `A` would be directly above point (25,90). And if `A` was a vertical label it would be plotted so that the left leg of the `A` would be directly to the left of the point (25,90).

Example: Earlier, we wrote a program to plot the men's and women's records for 100-meter freestyle swimming races in the Summer Olympics from 1948 through 1976. Now let's see how easy it is to label the graph. If you stored the program on page 219, load it now and add statements 55 and 3000 through 4000 of the following program:

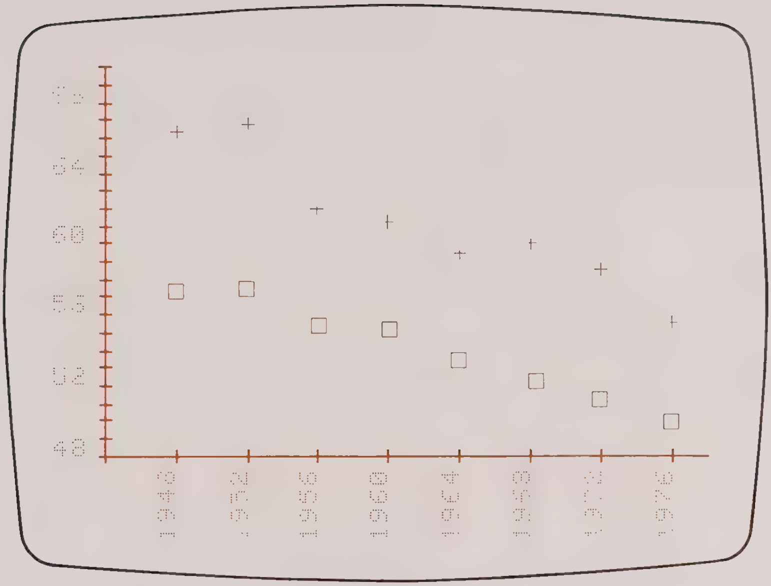


```
 10 REM *100-METER FREESTYLE SUM
    MER OLYMPICS 1948-1976
 20 GCLEAR
 30 SCALE 1940,1978,42,70.5
 40 XAXIS 48,4,1944,1978
 50 YAXIS 1944,1,48,70
 55 GOSUB 3000
 60 FOR I=1948 TO 1976 STEP 4
 70 READ M
 80 GOSUB 1000
 90 READ W
100 GOSUB 2000
110 NEXT I
120 DATA 57.3,66.3,57.4,66.8,55.
    4,62,55.2,61.2,53.4,59.5,52.
    2,60,51.22,58.59,49.99,55.65
130 STOP
1000 REM *PLOT SQUARE*
1010 MOVE I,M
1020 IMOVE -.2,.2
1030 IDRAW .4,0 @ IDRAW 0,-.4
1040 IDRAW -.4,0 @ IDRAW 0,.4
1050 RETURN
2000 REM *PLOT CROSS*
2010 MOVE I,W
2020 IMOVE 0,.3 @ IDRAW 0,-.6
2030 IMOVE .3,.3 @ IDRAW -.6,0
2040 RETURN
3000 REM *LABEL X-AXIS*
3010 LDIR 90
3020 FOR X=1948 TO 1976 STEP 4
3030 MOVE X,43
3040 LABEL VAL$(X)
3050 NEXT X
3060 REM *LABEL Y-AXIS*
3070 LDIR 0
3080 FOR Y=48 TO 70 STEP 4
3090 MOVE 1941, Y
3100 LABEL VAL$(Y)
3110 NEXT Y
3120 RETURN
4000 END
```

Go to the subroutine to label the axes.

Labels X-axis from 1948 through 1976 in increments of 4 years.
Moves to the designated X,Y position, then labels.

Labels Y-axis from 48 to 70 in increments of 4 seconds.

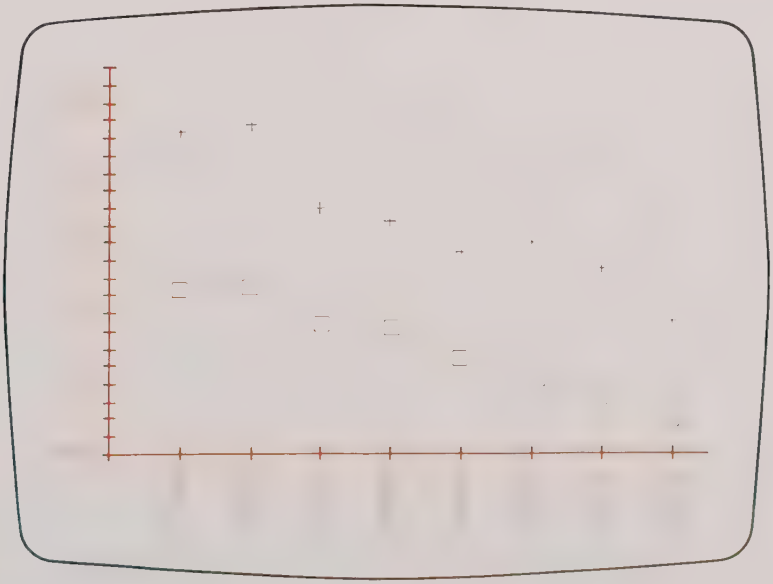


Horizontal labels are positioned immediately above the specified point; vertical labels are positioned immediately to the left of the specified point.

To center the labels next to the tic marks on the axes, change statements 3030 and 3090 to read as follows:

```
3030 MOVE X+.5,43
3090 MOVE 1941,Y-.5
```

Now run the program again:



Labels are centered on the tic marks.

You'll find that labels can be positioned easily at the desired location by adding or subtracting fractions of the units that you specify.

Of course, you could calculate the exact location of the label by scaling the graphics display to the number of plotting dots available, as we suggested earlier.

Problem

12.5 If you toss an unbiased coin a number of times, you will get all heads or all tails or more likely, some combination of heads and tails. Test your graphics programming skills by plotting a histogram of the theoretical probability distribution of the various numbers of heads you might obtain by tossing a fair coin ten times. Label the number of heads along the X-axis from 0 to 10. Since you will be graphing a histogram, center the labels under each unit on the axis. Label the probability along the Y-axis in intervals of 0.02 from 0 to 0.26.

Hints:

1. To find the various probabilities, evaluate each term of the binomial expansion:

$$(p+q)^n = \sum_{r=0}^{n} \frac{n!}{r!\,(n-r)!}\, p^{n-r} q^{r} \quad \text{where } p = q = 1/2 \text{ and } n = 10.$$

For example, to find the probability of obtaining three heads and seven tails in 10 tosses, evaluate the term:

$$\frac{10!}{3!\ 7!}\ (1/2)^7\ (1/2)^3 \approx 0.117$$

2. Define a factorial function to use in the above computation.
3. Remember to allow enough space in the `SCALE` statement for labels along the axes.

INPUT in Graphics Mode

One of the most useful features of HP-85 graphics is the system's ability to take inputs from the keyboard while the display remains in graphics mode. Thus, you can study the graphics display and input information to a program without the display reverting back to alpha mode.

There is an important difference between input in alpha mode and input in graphics mode. Whereas all of the display editing keys are active on input in alpha mode (e.g., the → key causes the cursor to move right, etc.), *only the* BACK SPACE *key is active on input in graphics mode*—the rest of the display editing keys will display their respective key-codes when pressed in response to input on the graphics display. Remember, the COPY, PAPER ADV, PAUSE, GRAPH, KEY LABEL, ROLL, and CLEAR keys are still active with graphics mode input. Refer to the table of key responses in appendix C.

Since the BACK SPACE key is the only editing feature allowed in graphics mode, it has been given some special capabilities. We'll discuss the backspace features in conjunction with graphics input in the following program.

Example: Write a program that generates 90 random numbers between 0 and 20 and plots them in order of generation on a horizontal scale from 0 to 30. Using `LABEL` statements on the graphics display, prompt for inputs for a graph heading, and for X- and Y-axis labels.

```
   5 REM *RANDOM DATA PLOT AND LA
     BEL
  10 DIM H$[32],Y$[14],X$[27]
  20 SCALE -5,30,-10,25
  30 PEN 1 @ GCLEAR
  40 XAXIS 0,1,0,30
  50 YAXIS 0,1,0,20
  60 RANDOMIZE
  70 FOR I=1 TO 90
  80 Y=20*RND
  90 PENUP
 100 PLOT I/3,Y
 110 NEXT I
 120 MOVE -5,-4
 130 LDIR 0
●140 LABEL "ENTER HEADING"
 150 MOVE -5,-6
●160 INPUT H$
 170 H=LEN(H$)
 180 H1=10-INT(H/2)*30/32
 190 MOVE H1,22
 200 LABEL H$
 210 GCLEAR -1
 220 MOVE -5,-4
●230 LABEL "LABEL Y-AXIS (MAX 14
     CHARS)"
 240 MOVE -5,-6
●250 INPUT Y$
 260 Y=LEN(Y$)
 270 Y1=10-INT(Y/2)*10/7
 280 MOVE -1,Y1
 290 LDIR 90
 300 LABEL Y$
 310 GCLEAR -1
 320 MOVE -5,-6
 330 LDIR 0
●340 LABEL "NOW LABEL X-AXIS (MAX
     27 CHARS)"
 350 MOVE -5,-8
●360 INPUT X$
 370 X=LEN(X$)
 380 X1=12-INT(X/2)*32/35
 390 GCLEAR -2
 400 MOVE X1,-3
 410 LABEL X$
 420 END
```

Dimensions string input variables.
Scales X and Y units.

Draws axes.

Plots random points between 0 and 20 in order of generation.
Moves to desired point, then labels.

Moves to desired input point on graphics display.

Centers heading.

Then labels.
Clears previous prompt and input from graphics display.
Prompts for next input.

Moves to desired point of question mark appearance.

Centers label along Y-axis.

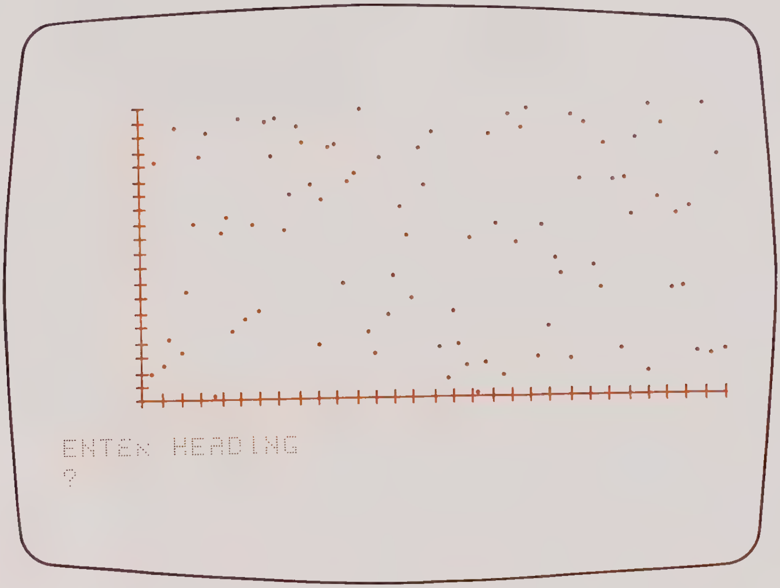
Changes label direction.
Labels Y-axis.
Clears previous prompt and input from graphics display.

Inputs X-axis label.

Centers label under X-axis.
Clears prompt and input.

Then labels.

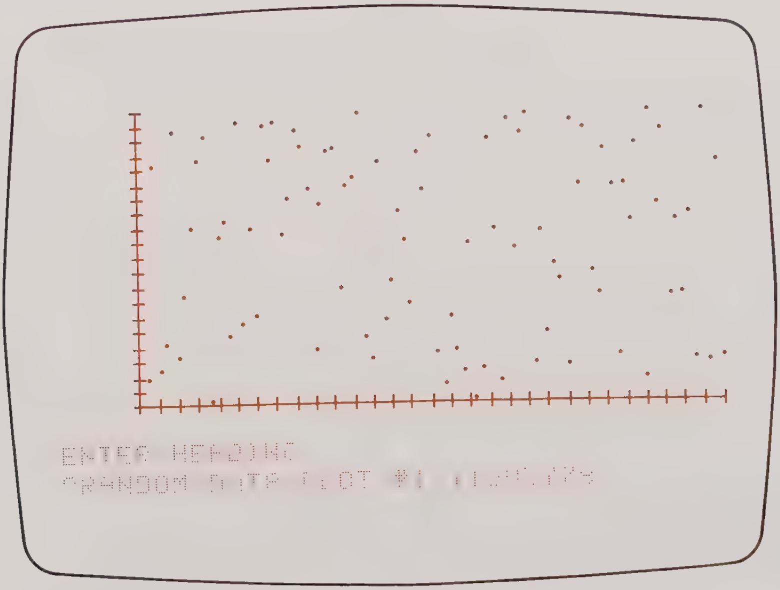
After you have entered the program, press (RUN). The graphics display shows:



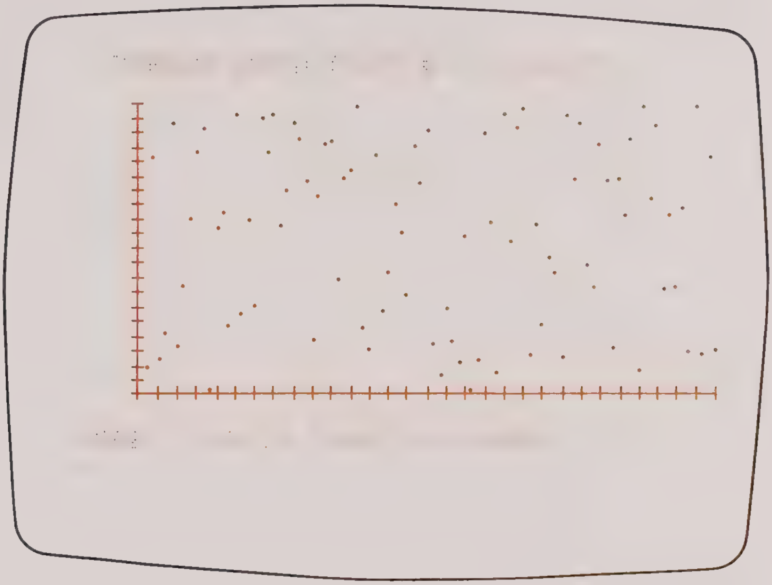
The input prompt, ?, appears on the graphics display if an INPUT statement is executed when the CRT display is in graphics mode.

Since you moved to point −5, −6 on the graphics display before the input statement, the question mark appears at point −5, −6. Now enter a heading for the data plot. If you make a typing mistake, backspace to erase and correct the error.

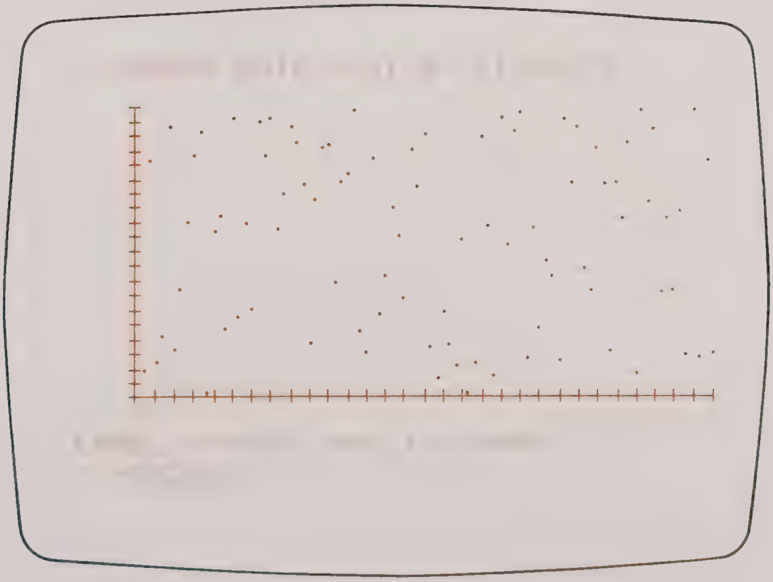
When you have completed typing the heading, press (END LINE). The display will remain in graphics mode while you type.



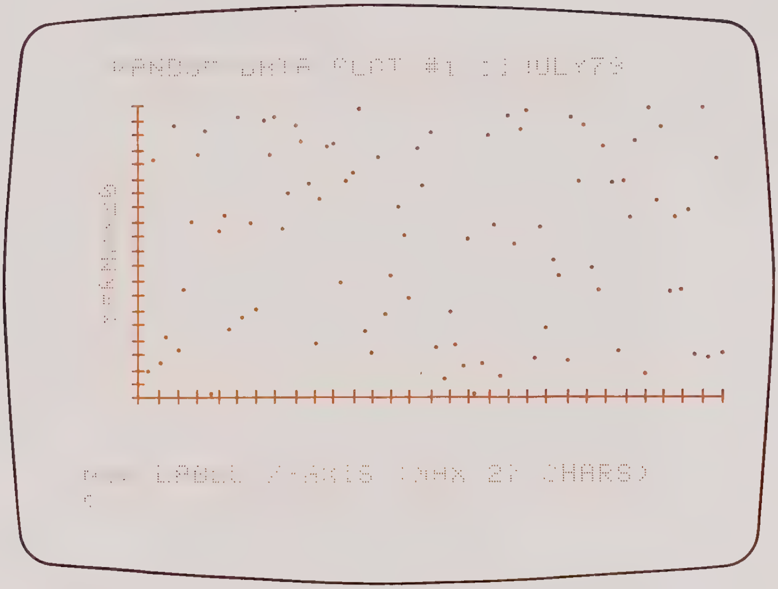
After you enter the heading, the program will prompt for the next label.



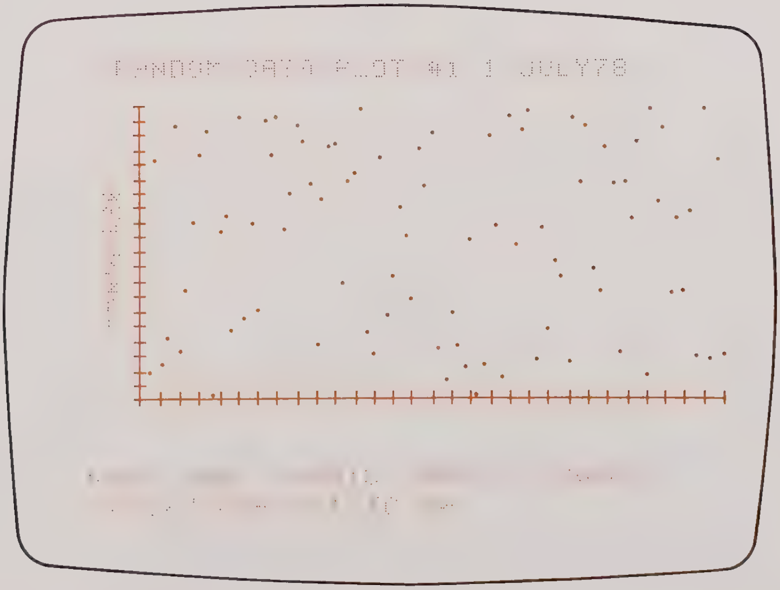
Again, the question mark appeared at point −5, −6 because we moved to that point prior to the `INPUT` statement. Enter the Y-axis label and press END LINE.



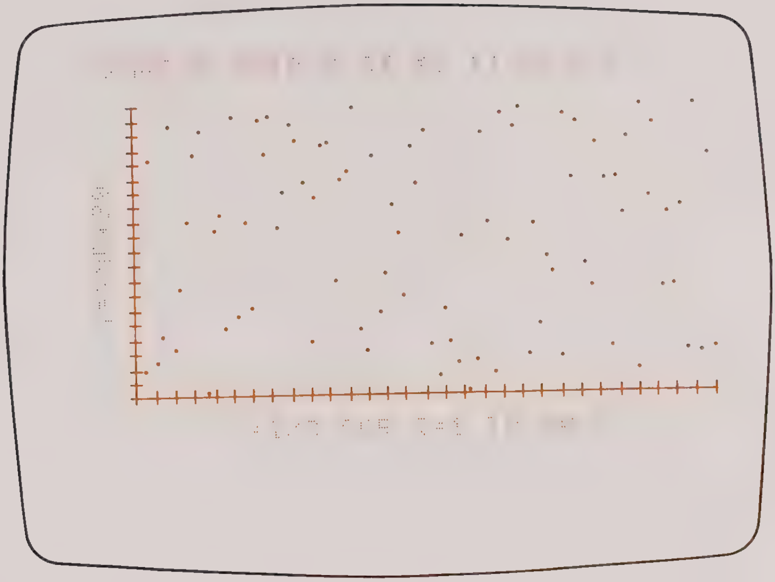
Now the program prompts for the last label.


RANDOM DATA PLOT #1 1 JULY78
ENTER LABEL Y-AXIS (MAX 27 CHARS)

Enter the label and press (END LINE).


RANDOM DATA PLOT #1 1 JULY78

The program centers the last input under the X-axis.

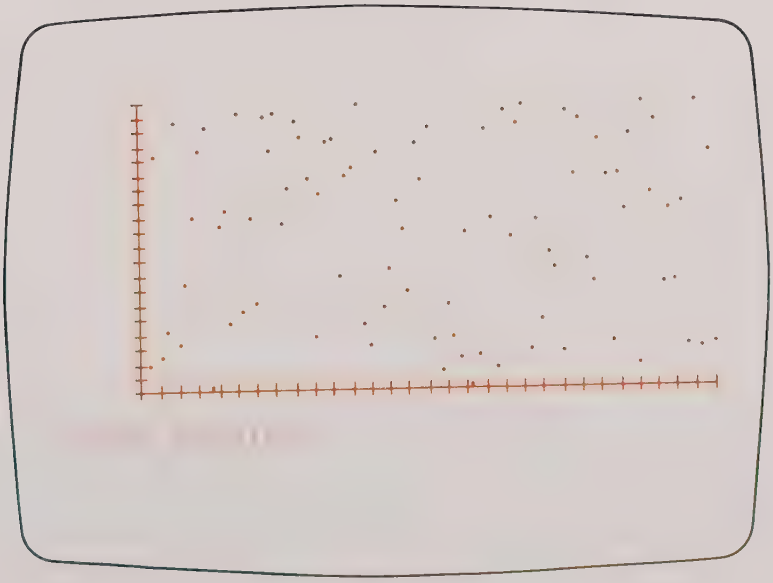


Experiment with the position of the input prompt, `?`, to view the results of inputting information to the graphics display.

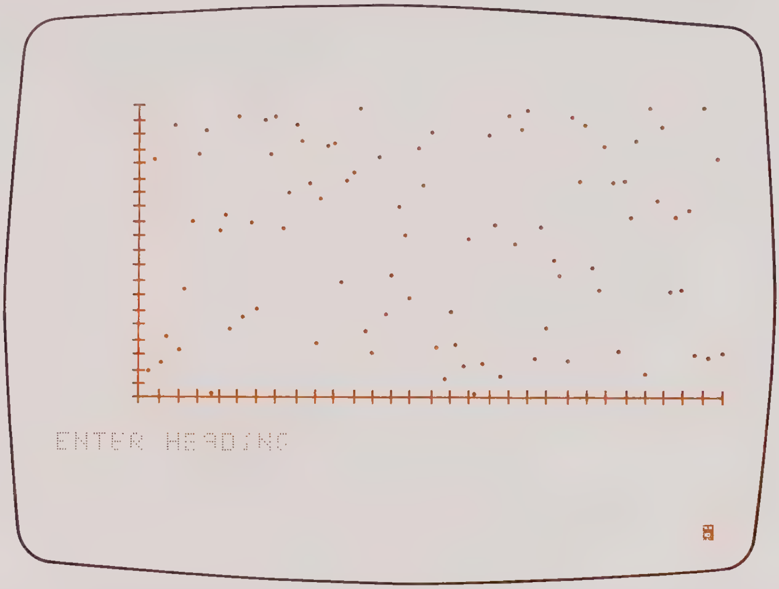
For instance, if you change statement 150 to:

```
150 MOVE 29,-10
```

The first input prompt will be displayed in the lower right corner.



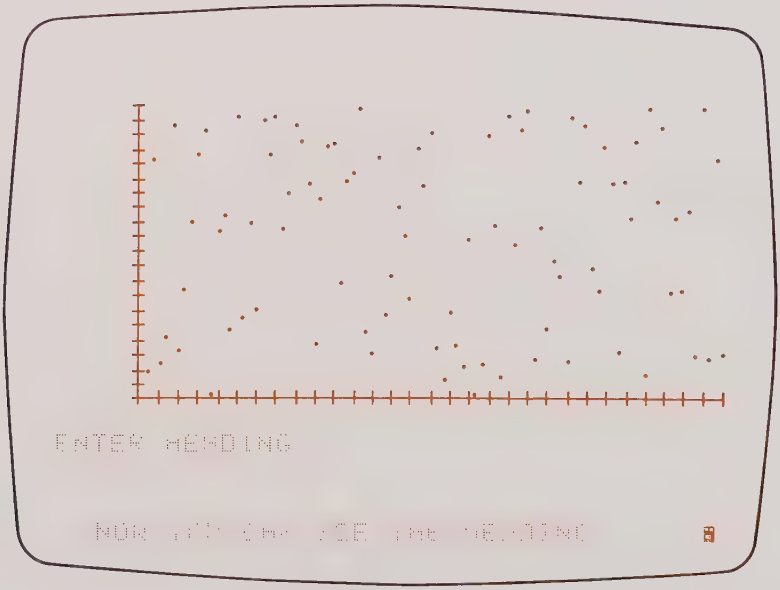
You can enter up to 95 characters (in our program 32 characters) in an input statement, but you won't be able to see what you have entered since the graphics display does not scroll up like the alpha display when characters are typed. Thus, the message would be typed on top of itself in the lower right corner.



```
ENTER HEADING
```

Even though you cannot distinguish the characters that have been keyed in, the system remembers up to 95 characters. So, you can still backspace to correct a character if you've made a mistake.

Since there is space on the graphics display to backspace, the system allows you to backspace past the question mark. After you backspace, you can enter the desired input and view the message as you input it.



```
ENTER HEADING

NOW YOU CAN SEE THE HEADING
```

An `INPUT` statement in graphics mode always resets the label direction to the horizontal position so that input messages can be read easily. The input prompt, `?`, always appears on the graphics display if the CRT is in graphics mode when an `INPUT` statement is executed. If you really want to input in alpha mode, be sure to execute the `ALPHA` statement prior to the `INPUT` statement.

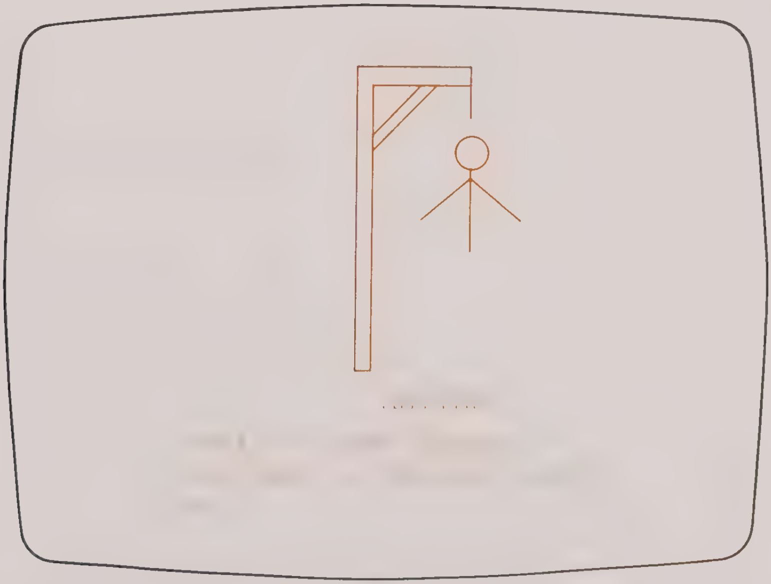
Problem

12.6 "Hangman" is a game commonly played by youngsters (and oldsters, alike) in which one person chooses a word and another must guess it, one letter at a time, given the length of the word. The word-chooser writes a dash to represent each letter in the word. Whenever a letter is guessed correctly, all occurrences of the letter in the word are written above the dash that represents the letter's position in the word. Whenever an incorrect guess is made, a part of the hangman's body is drawn.

If the word is guessed before the hangman is completed, the guesser wins. If the hangman is completed before the word is discovered, the word-chooser wins.

Write a program to simulate this game on the HP-85. So that you do not have to create string data files, write it in such a way that one person inputs a word, the display is cleared, and another person must guess the word. Write one subroutine to draw the scaffold and another subroutine that includes a computed `GOTO` statement to determine the part of the body that is to be drawn in the case of an incorrect guess. Include six body parts (head, left and right arms, trunk, left and right legs) and allow the guesser six incorrect guesses.

Here's a sample graphics display where the guesser won with two guesses left and the hangman body 2/3 complete.



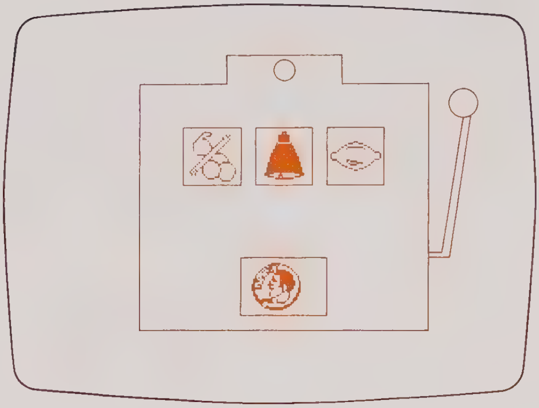
With the hangman program, it is essential to accept inputs in graphics mode.

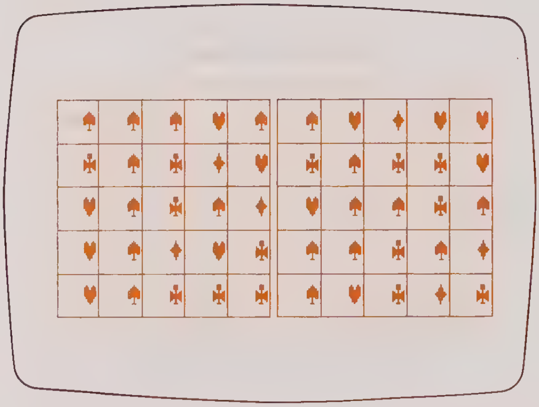
Advanced Plotting With BPLOT

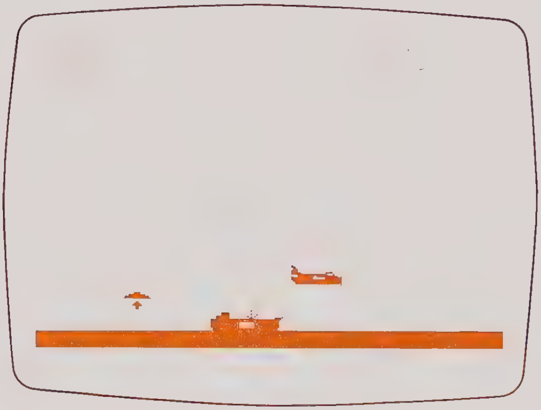
The `BPLOT` *(byte plot)* statement enables you to plot groups of dots on the graphics display by creating a string of characters that specify those dots. Each character in the string specifies one byte (eight bits or dots) of information which determines whether dots are on or off on the graphics display.

`BPLOT` string expression , number of characters per row

The BPLOT statement is not difficult to use, but it does take some time to figure out the precise dot configurations of a design or pattern. If you have played any of the games in the HP-85 Games Pac, you've probably seen BPLOT in action. Soon you'll be generating figures like these using BPLOT:







First we will outline the procedure for building a character string for BPLOT, then we will discuss some examples and byte plotting peculiarities.

Procedure for Building the String

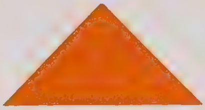
1. Draw the figure you wish to plot.
2. Then redraw the figure in matrix form, using dot patterns instead of lines. Graph paper is useful at this point; let each square equal one dot, block, or bit of information.
3. Divide the dot figure into columns of dots and spaces, eight squares wide. View each eight blocks as a byte of information where each block specifies a bit. If a dot is specified, the value of the block is one; if no dot is specified, the block's value is zero. Thus, each group of eight dots or spaces specifies a binary number that determines a particular character.
4. Convert each binary number to its decimal equivalent. This can be done in a variety of ways; the easiest of course, is to use a conversion table. (You can use the table of characters and binary/octal/decimal equivalents in appendix C.) If a table is not available, convert each eight-digit binary number to its three-digit octal equivalent and then convert the octal number to its decimal equivalent (since most of us don't easily convert binary numbers to decimal equivalents). You may wish to use (or modify) the Base Conversions program, listed on page 257.

5. Build the character string by assigning the character of the specified decimal value (using the `CHR$` function) to the appropriate character position in the string; the easiest way to build the string is to write a program that accepts and appends the character to the string through `INPUT` statements or `READ` and `DATA` statements.
6. Use this string with the `BPLOT` statement to plot the figure. Examples of the statement are:

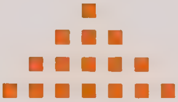
`BPLOT T$,1`	Plots T$; 1 character per row of dots.
`BPLOT S$,5`	Plots S$; 5 characters per row of dots.

Let's take a simple example to illustrate the first five steps of the procedure. Suppose you wish to plot a solid triangle:

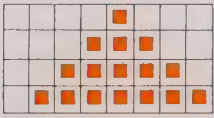
Step 1. Draw the figure.



Step 2. Represent the figure with dots or blocks.

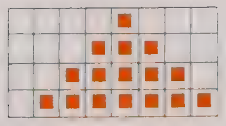


Step 3. Since the base of the triangle is only seven dots wide we need only to place it in a four by eight dot matrix.



Each row of this dot matrix specifies a byte (eight bits) of information.

Step 4. Convert each row of the matrix to a decimal value.



Binary Representation	**Octal Value**	**Decimal Value**
00 001 000	0 1 0	8
00 011 100	0 3 4	28
00 111 110	0 7 6	62
01 111 111	1 7 7	127

Step 5. Build the string using the `CHR$` function:

```
T$=CHR$(8)&CHR$(28)&CHR$(62)&CHR
$(127)
T$
△œ>├
```

This is a short string so we have manually entered the characters. To see the string, type the variable name, then press END LINE.

You should use the `CHR$` function to build this string since some characters cannot be specified from the keyboard (e.g., those with decimal values over 128), and others have special meanings when found in association with strings (e.g., the quotation mark).

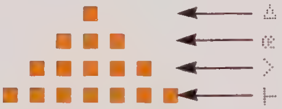
Since T$ is a short string, we built it from the keyboard. With longer strings, you might write a program like this:

```
10 DIM T$[4]
20 FOR I=1 TO 4
30 READ V
40 T$[I,I]=CHR$(V)
50 NEXT I
60 DATA 8,28,62,127
70 END
```

Dimensions the variable.
Uses a `FOR-NEXT` loop to `READ` or `INPUT` the decimal values into the appropriate character position in the string.

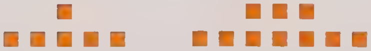
Step 6. Use the string with the `BPLOT` statement to plot the figure. Below we have enlarged the graphics display area around each `BPLOT` to illustrate the statement. Do not execute these statements now.

```
BPLOT T$,1
```



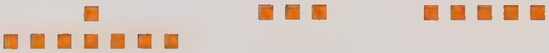
Plots one character per line of `T$="△œ>├"`, thus producing a triangle.

```
BPLOT T$,2
```



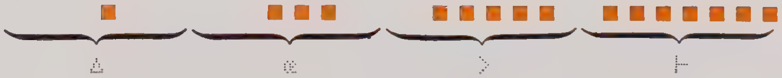
Plots two characters per line of `T$="△œ>├"`.

```
BPLOT T$,3
```



Plots three characters per line of `T$="△œ>├"`.

```
BPLOT T$,4
```



Plots four characters per line of `T$="△œ>├"`.

As you can see, `BPLOT T$,1` produces a triangle because it plots one character per line. `BPLOT T$,4` plots all four characters on the same line.

Using the String With BPLOT

Now that you have composed the string, use the BPLOT statement to plot the figure. Enter and run the following program—use the editing features of your HP-85 to add statements to the last program if you wish and then renumber the program.

```
 10 PEN 1 @ GCLEAR
 20 SCALE 0,255,0,191

 30 FOR I=1 TO 4
 40 READ V
 50 T$[I,I]=CHR$(V)
 60 NEXT I
 70 MOVE 124,100
 80 FOR I=1 TO 11
 90 BPLOT T$,1
100 NEXT I
110 FOR X=100 TO 148 STEP 8
120 MOVE X,80
130 BPLOT T$,1
140 NEXT X
150 DATA 8,28,62,127
160 END
```

Clears the graphics display.

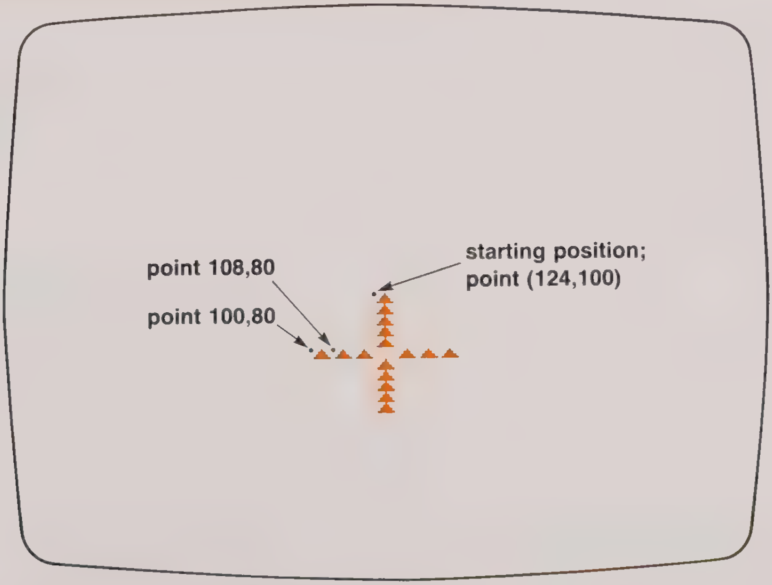
Scales to number of dots on graphics screen.

Repeats the procedure for building the string.

Moves to point 124,100.

Creates a column of 11 triangles.

Creates a row of seven triangles.



Note:

1. BPLOT automatically stacks the specified string when only one pen location is specified (see lines 70 through 100 above).
2. BPLOT performs an EXOR (exclusive or) with existing dots on the screen. Thus, we erased the middle triangle by plotting it twice.

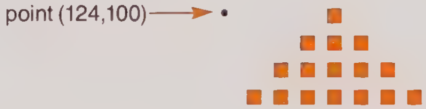
The example above illustrates most of the facts you need to know about BPLOT. We enumerate them here:

1. For your ease in using BPLOT, scale the display from 0 to 255 (256 dots in the horizontal direction) and from 0 to 191 (192 dots in the vertical direction). With this scale, you always know exactly where the dots will be plotted.

2. The starting position of a byte plot always has an X-coordinate value that is a multiple of four on a horizontal scale of 0 to 255.

   If the current pen location does not have an X-coordinate 0,4,8,...252, the figure will be justified to the nearest four-dot position (multiple of four using the scale above) to the left of the current pen location. The figure is plotted with the upper left corner at the specified pen position. In our example, the statements on the left produced the figure on the right.

```
70 MOVE 124,100
90 BPLOT T$,1
```



3. If the BPLOT statement is executed several times without changing the original pen location, the second figure is plotted immediately below the first figure, the third below the second, etc. Notice that lines 70 through 100 produced a column of 11 triangles with the upper left bit of the first triangle at point 124,100. When more than one BPLOT statement is executed, one after the other, it's as if the graphics display performs a carriage-return and tabs to the original horizontal position, to begin plotting the figure immediately below the first.
4. BPLOT performs an EXOR (exclusive or) operation between the character string you specify and the existing dots on the display. As you have seen, the middle triangle above was erased because we plotted it twice. Let's discuss how this occurred.

The table below illustrates all possible conditions and outcomes of the EXOR operation between a dot on the screen and the same dot specified by a BPLOT string. The third column gives the resulting dot condition; 0 means the dot is off and 1 means the dot is on.

Dot before BPLOT	Same dot specified by BPLOT string	EXOR: Resultant dot condition
1 (on)	1	0 (off)
1 (on)	0	1 (on)
0 (off)	1	1 (on)
0 (off)	0	0 (off)

If a dot is on, specifying 1 for the same dot in the BPLOT string turns the dot off.
If the dot is on, 0 in the string keeps it on.

If the dot is off, 1 turns it on.

If the dot is off, 0 keeps it off.

Here's what happened to the middle triangle:

GCLEAR point(124,80)	first BPLOT T$,1 at point (124,80)	display after second BPLOT T$,1 at point (124,80)
0 0 0 0 0 0 0 0	0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 1 1 1 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 1 1 1 1 1 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0
GCLEAR with PEN 1 turns all dots off.	Since 0 EXOR 1 = 1 and 0 EXOR 0 = 0; plots triangle.	Since 1 EXOR 1 = 0 and 0 EXOR 0 = 0; erases triangle.

This aspect of BPLOT becomes very important when you wish to simulate a figure moving across the graphics display. You must know whether a 0 or a 1 will turn the dot on or off of the current display.

Condensing the String Assignment Program

Once you have defined the string, as we have done in the last two programs, you can create one assignment statement that specifies the string for use in future programs. This will shorten your programs by at least four or five statements. More important, for complicated figures, it will eliminate the long set-up time.

For instance, while the last program (page 241) is still in computer memory, type:

```
DISP "30 T$="&CHR$(34)&T$&CHR$(3
4)
```

Execute the statement, by pressing END LINE, to display:

```
30 T$="⊥æ>⊢"
```

Now use the ↑ key to move the cursor back to this line and then press END LINE; the new statement 30 will be stored.

Notice that we have used `CHR$(34)` to specify quotes. If your `BPLOT` character string also contains quotes, you must concatenate them to the string using `CHR$(34)`.

If your `BPLOT` string contains underlined characters (characters with decimal values above 128), you must be careful to avoid the underlined character with the cursor. The cursor will always erase an underline, thus changing the character value.

Since the new statement 30 has been stored, you can delete the unnecessary statements from the program. Do so now by executing:

`DELETE 40,60` END LINE	Removes rest of `FOR-NEXT` loop.
`150` END LINE	Deletes `DATA` statement.

Renumber the program and list it on the display:

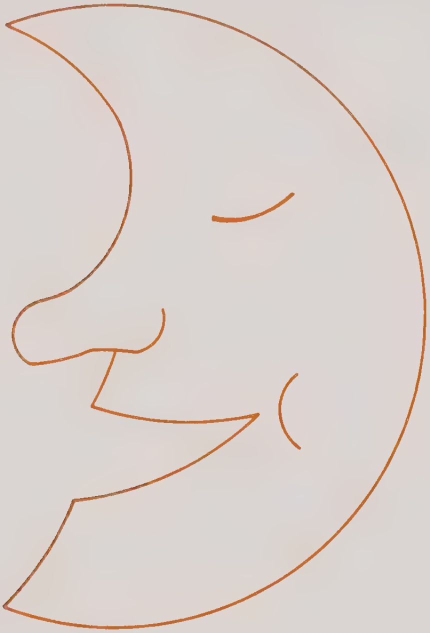
```
10 PEN 1 @ GCLEAR
20 SCALE 0,255,0,191
30 T$="⊥æ>⊢"
40 MOVE 124,100
50 FOR I=1 TO 11
60 BPLOT T$,1
70 NEXT I
80 FOR X=100 TO 148 STEP 8
90 MOVE X,80
100 BPLOT T$,1
110 NEXT X
120 END
```

This program, shortened by four program lines, performs exactly the same `BPLOT` as the program on page 241. Try it! It will also be faster for longer strings!

Let's look at a more difficult example to illustrate the remaining features of the `BPLOT` statement.

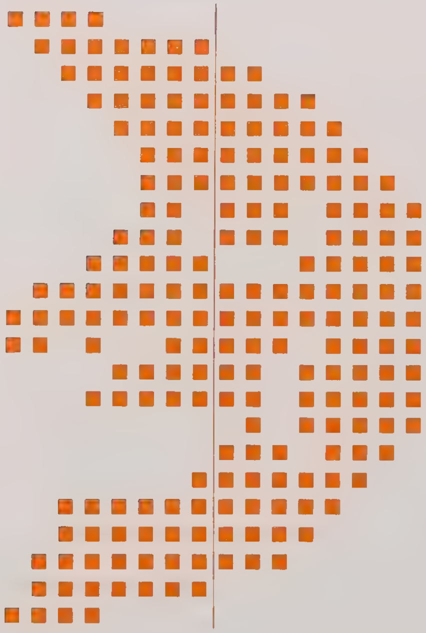
Example: Use `BPLOT` to plot a man in the moon. Then create another `BPLOT` character string to make the man in the moon move across the screen, one byte each move.

1. First draw the figure:



2. Now redraw the figure on graph paper in matrix form. Since the moon is white, we want to light all of the dots inside the figure except the lines drawn for the eye, nose, and cheek.

Matrix Form



Binary Representation

```
1 1 1 1 0 0 0 0|0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1|0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1|1 1 0 0 0 0 0 0
0 0 0 1 1 1 1 1|1 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1|1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1|1 1 1 1 1 1 0 0
0 0 0 0 0 1 1 1|1 1 1 1 1 1 1 0
0 0 0 0 0 1 1 0|1 1 1 0 1 1 1 1
0 0 0 0 1 1 1 0|1 1 1 0 1 1 1 1
0 0 0 1 1 1 1 1|0 0 0 1 1 1 1 1
0 1 1 1 1 1 1 1|1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1|1 1 1 1 1 1 1 1
1 1 0 1 0 0 1 1|1 1 1 0 1 1 1 1
0 0 0 0 1 1 1 1|1 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1|1 1 0 1 1 1 1 1
0 0 0 0 0 0 0 0|0 1 0 1 1 1 1 1
0 0 0 0 0 0 0 0|1 1 1 0 1 1 1 0
0 0 0 0 0 0 0 1|1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1|1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1|1 1 1 1 0 0 0 0
0 1 1 1 1 1 1 1|1 1 1 0 0 0 0 0
0 1 1 1 1 1 1 1|0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0|0 0 0 0 0 0 0 0
```

3. Divide the figure into columns of dots and spaces, eight squares wide. Our figure is 16 squares wide, so we divided it into two columns. Each line of each column represents one byte of information. Above, we also converted the figure to its binary representation. For the two columns of eight squares, we also have two columns of eight-digit binary numbers.

4. Convert each eight-digit binary number to its decimal equivalent.

Binary Representation		Octal Representation		Decimal Value	
11 110 000	00 000 000	360	000	240	0
01 111 111	00 000 000	177	000	127	0
00 111 111	11 000 000	077	300	63	192
00 011 111	11 110 000	037	360	31	240
00 001 111	11 111 000	017	370	15	248
00 000 111	11 111 100	007	374	7	252
00 000 111	11 111 110	007	376	7	254
00 000 110	11 101 111	006	357	6	239
00 001 110	11 101 111	016	357	14	239
00 011 111	00 011 111	037	037	31	31
01 111 111	11 111 111	177	377	127	255
11 111 111	11 111 111	377	377	255	255
11 010 011	11 101 111	323	357	211	239
00 001 111	11 011 111	017	337	15	223
00 011 111	11 011 111	037	337	31	223
00 000 000	01 011 111	000	137	0	95
00 000 000	11 101 110	000	356	0	238
00 000 001	11 111 100	001	374	1	252
00 111 111	11 111 000	077	370	63	248
00 111 111	11 110 000	077	360	63	240
01 111 111	11 100 000	177	340	127	224
01 111 111	00 000 000	177	000	127	0
11 110 000	00 000 000	360	000	240	0

5. Build the character string using the `CHR$` function.

```
10 REM *BUILD MOON STRING*
20 DIM M$[46]

30 FOR I=1 TO 46
40 READ M1
50 M$[I,I]=CHR$(M1)
60 NEXT I
70 DATA 240,0,127,0,63,192,31,2
   40,15,248,7,252,7,254,6,239,
   14,239,31,31,127,255,255,255
   ,211,239
80 DATA 15,223,31,223,0,95,0,23
   8,1,252,63,248,63,240,127,22
   4,127,0,240,0
90 END
```

Dimensions string to number of decimal values.

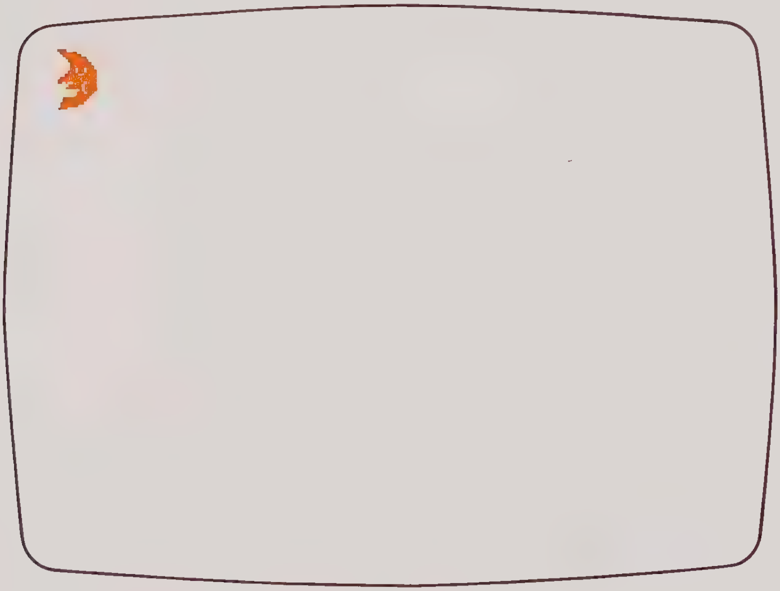
Uses `FOR-NEXT` loop to `READ` or `INPUT` the decimal values and assigns them to the appropriate position in the character string.

Data for moon string read from decimal value table from left to right.

6. Use this string with the `BPLOT` statement to plot the man in the moon. Append the following statements to the end of the string building program above and then press (RUN).

```
 90 SCALE 0,255,0,191
100 PEN 1 @ GCLEAR
110 MOVE 0,191
120 BPLOT M$,2
130 END
```

Replace `END` statement with `SCALE` statement.
Moves to upper left corner of display.
`BPLOT` the character string, two characters per line.



As you can see, the man in the moon was plotted once in the upper left corner of the graphics display.

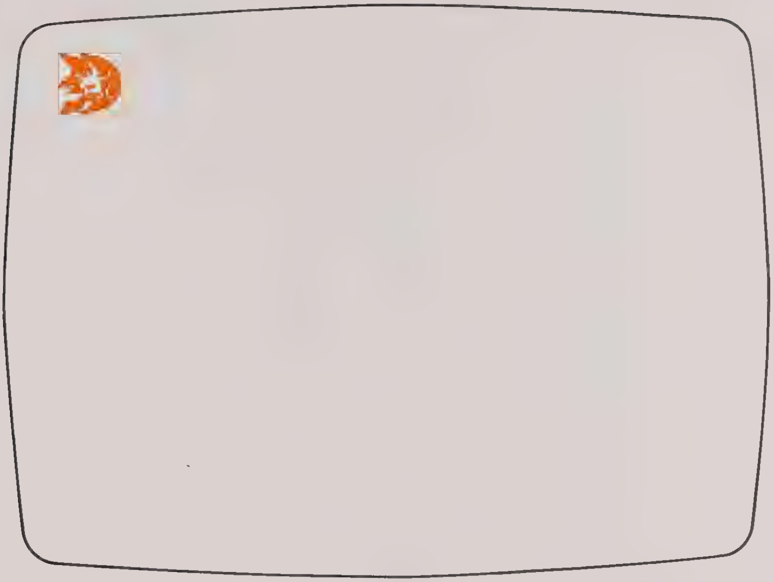
To finish the solution to the example, we must move the man in the moon across the display one byte (eight dots) at a time.

What happens when we simply position the pen to point 8,191—eight dots from the original starting position, and then execute `BPLOT M$,2` once again? Try it!

```
MOVE 8,191


BPLOT M$,2
```

Moves to point 8,191. Remember the system reverts to alpha mode as you type a statement and reverts back to graphics mode when you execute it.
Byte plots the character string for the moon, two characters per line.

After you execute the BPLOT statement, the display shows:



As you can see, BPLOT performs an EXOR operation with existing dots on the graphics display. So, the left half of the first moon remains intact, but the right half of the first moon and the left half of the second moon leave an odd dot configuration on the display. Since the display was clear to begin with (aside from the first moon), the right half of the second moon is plotted correctly.

This should give you an idea of what we must do in order to simulate the moon moving across the display. We must create another character string for BPLOT—three bytes (characters) wide. The first character should erase the left half of the first moon, the second character should plot the left half of the second moon when it is plotted on top of the right half of the first moon, and the third character should plot the right half of the second moon.

The first and third characters are easy enough to compute. Since BPLOT performs an EXOR with existing dots on the display, the first character of each line of our new BPLOT string is the same as the first character of the original string; 1EXOR1=0 and 0EXOR0=0. The third character of each line of the new string is the same as the second character of each line in the original string; 0EXOR1=1 and 0EXOR0=0.

The middle character of each line of our new BPLOT must be computed such that it produces the left half of the moon. Since it is plotted on top of the first moon, you must specify the bit value 0 or 1 so that when an EXOR is performed, you obtain the desired result.

If a dot is on and you want it off, specify 1.
If a dot is off and you want it on, specify 1.
If a dot is on and you want it on, specify 0.
If a dot is off and you want it off, specify 0.

In other words, the middle character is an EXOR between the first half and the second half of the original moon.

Binary Representation of New Moon

Left Half	EXOR	Right Half
1 1 1 1 0 0 0 0	1 1 1 1 0 0 0 0	0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 0 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 0 1 1 1 1	1 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1	1 1 1 1 0 1 1 1	1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1	1 1 1 1 1 0 1 1	1 1 1 1 1 1 0 0
0 0 0 0 0 1 1 1	1 1 1 1 1 0 0 1	1 1 1 1 1 1 1 0
0 0 0 0 0 1 1 0	1 1 1 0 1 0 0 1	1 1 1 0 1 1 1 1
0 0 0 0 1 1 1 0	1 1 1 0 0 0 0 1	1 1 1 0 1 1 1 1
0 0 0 1 1 1 1 1	0 0 0 0 0 0 0 0	0 0 0 1 1 1 1 1
0 1 1 1 1 1 1 1	1 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1
1 1 0 1 0 0 1 1	0 0 1 1 1 1 0 0	1 1 1 0 1 1 1 1
0 0 0 0 1 1 1 1	1 1 0 1 0 0 0 0	1 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1	1 1 0 0 0 0 0 0	1 1 0 1 1 1 1 1
0 0 0 0 0 0 0 0	0 1 0 1 1 1 1 1	0 1 0 1 1 1 1 1
0 0 0 0 0 0 0 0	1 1 1 0 1 1 1 0	1 1 1 0 1 1 1 0
0 0 0 0 0 0 0 1	1 1 1 1 1 1 0 1	1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1	1 1 0 0 0 1 1 1	1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1	1 1 0 0 1 1 1 1	1 1 1 1 0 0 0 0
0 1 1 1 1 1 1 1	1 0 0 1 1 1 1 1	1 1 1 0 0 0 0 0
0 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0	1 1 1 1 0 0 0 0	0 0 0 0 0 0 0 0

The binary numbers in this column are the same as the binary numbers in the first column of the original moon.

The binary numbers in the middle column are the results of the EXOR operation performed between digits of the left half and the corresponding digits of the right half of the moon.

The binary numbers in this column are the same as the binary numbers in the second column of the original moon.

We already know the decimal values of the first and third column. Now find the decimal values of the numbers in the middle column.

Binary Representation	Octal Representation	Decimal Value
1 1 1 1 0 0 0 0	3 6 0	2 4 0
0 1 1 1 1 1 1 1	1 7 7	1 2 7
1 1 1 1 1 1 1 1	3 7 7	2 5 5
1 1 1 0 1 1 1 1	3 5 7	2 3 9
1 1 1 1 0 1 1 1	3 6 7	2 4 7
1 1 1 1 1 0 1 1	3 7 3	2 5 1
1 1 1 1 1 0 0 1	3 7 1	2 4 9
1 1 1 0 1 0 0 1	3 5 1	2 3 3
1 1 1 0 0 0 0 1	3 4 1	2 2 5
0 0 0 0 0 0 0 0	0 0 0	0
1 0 0 0 0 0 0 0	2 0 0	1 2 8
0 0 0 0 0 0 0 0	0 0 0	0
0 0 1 1 1 1 0 0	0 7 4	6 0
1 1 0 1 0 0 0 0	3 2 0	2 0 8
1 1 0 0 0 0 0 0	3 0 0	1 9 2
0 1 0 1 1 1 1 1	1 3 7	9 5
1 1 1 0 1 1 1 0	3 5 6	2 3 8
1 1 1 1 1 1 0 1	3 7 5	2 5 3
1 1 0 0 0 1 1 1	3 0 7	1 9 9
1 1 0 0 1 1 1 1	3 1 7	2 0 7
1 0 0 1 1 1 1 1	2 3 7	1 5 9
0 1 1 1 1 1 1 1	1 7 7	1 2 7
1 1 1 1 0 0 0 0	3 6 0	2 4 0

Thus, the decimal values for our second `BPLOT` character string are:

	Decimal Values	
1st Character	2nd Character	3rd Character
240	240	0
127	127	0
63	255	192
31	239	240
15	247	248
7	251	252
7	249	254
6	233	239
14	225	239
31	0	31
127	128	255
255	0	255
211	60	239
15	208	223
31	192	223
0	95	95
0	238	238
1	253	252
63	199	248
63	207	240
127	159	224
127	127	0
240	240	0

Finally, build the character string for the second moon. With the previous moon program still intact in computer memory, make the following changes:

1. Dimension the second moon for 69 characters in statement 20.
2. Add statements 125 through 200, below, to build the second string, and statements 210 to 9999 to plot the string.

```
 10 REM *BUILD MOON STRING*
 20 DIM M$[46],M2$[69]
 30 FOR I=1 TO 46
 40 READ M1
 50 M$[I,I]=CHR$(M1)
 60 NEXT I
 70 DATA 240,0,127,0,63,192,31,2
    40,15,248,7,252,7,254,6,238,
    14,239,31,31,127,255,255,255
    ,211,239
 80 DATA 15,223,31,223,0,95,0,23
    8,1,252,63,248,63,240,127,22
    4,127,0,240,0
 90 SCALE 0,255,0,191
100 PEN 1 @ GCLEAR
110 MOVE 0,191
120 BPLOT M$,2
125 REM *BUILD 2nd MOON*
130 FOR K=1 TO 69
140 READ M2
```

Dimensions the string variable for the second moon, M2$.

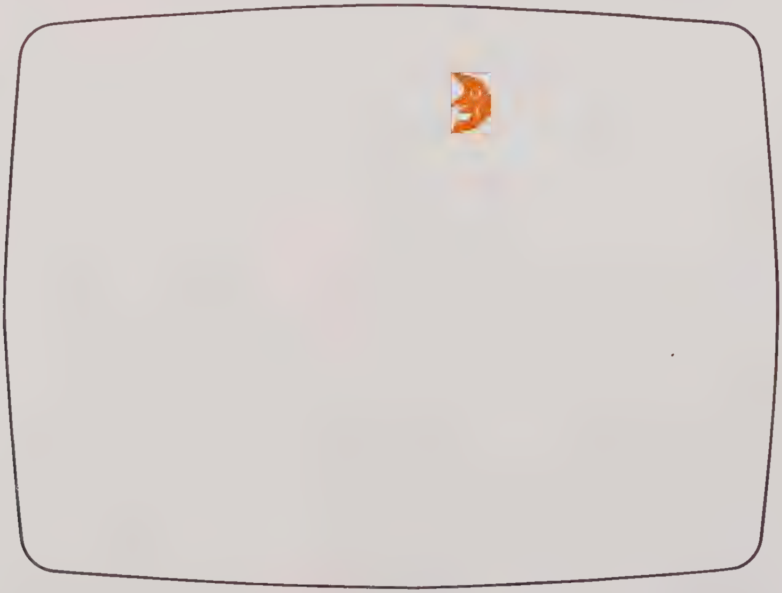
Builds the second string using `DATA` statements 170 through 200.

```
150 M2$[K,K]=CHR$(M2)
160 NEXT K
170 DATA 240,240,0,127,127,0,63,
    255,192,31,239,240,15,247,24
    8,7,251,252
180 DATA 7,249,254,6,233,239,14,
    225,239,31,0,31,127,128,255,
    255,0,255,211,60,239
190 DATA 15,208,223,31,192,223,0
    ,95,95,0,238,238,1,253,252,6
    3,199,248
200 DATA 63,207,240,127,159,224,
    127,127,0,240,240,0
210 REM *NOW SET UP LOOPS SO THE
        MOON MOVES FROM LEFT
        TO RIGHT ACROSS DISPLAY
220 REM AND FROM TOP TO BOTTOM
        RIGHT CORNER*
230 FOR Y=191 TO 0 STEP -1

240 FOR X=0 TO 255 STEP 8
250 MOVE X,Y
260 BPLOT M2$,3
270 WAIT 1000
280 NEXT X
290 NEXT Y
9999 END
```

Outer loop (*Y*) moves string from top to bottom.
Inner loop (*X*) moves string from left to right.

Wait statement pauses to display the moon in its current location.

Now run the program to see the man in the moon move across the display from left to right.



We chose to move the moon eight dots at a time in the horizontal direction. But we could have moved any number of dots at a time. For instance, if we had moved it four dots at a time, the outer four squares of the second moon would be the same as the first. But each of the middle four dots would have to be computed as the `EXOR` of that set of four dots and the four dots preceding it.

You can stop the moon program anytime by pressing PAUSE.

Finally, you may wish to condense the moon program in the same manner that we condensed the triangle program earlier.

Since you have just executed the moon program, variable M$ contains character data to build the first moon and variable M2$ contains character data to build the moving moon.

After pressing (PAUSE) to stop the program, create one assignment statement for each moon variable.

Create an assignment statement for M$ by executing:

```
DISP "1000 M$="&CHR$(34)&M$&CHR$
(34)
```

When you execute this statement, an assignment statement numbered 1000 will appear on the display.

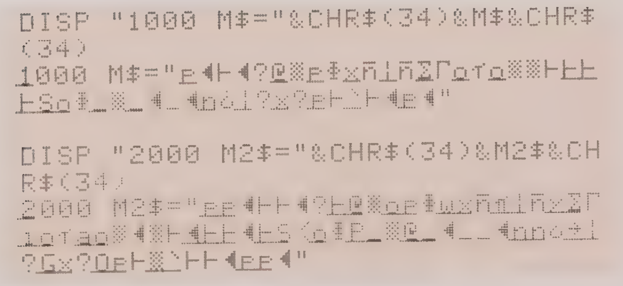
Do not enter the new assignment statement into computer memory until you have displayed the characters of each variable you wish to enter. In other words, do not press (END LINE) after statement 1000—the statement you've just created—now. Doing so would deallocate all program variables so that M2$ would be undefined once again.

First, create an assignment statement for M2$ by executing:

```
DISP "2000 M2$="&CHR$(34)&M2$&CH
R$(34)
```

When you executed the statements above, the system displayed the character strings composing each variable. Now you must enter the program lines that you have created into computer memory. Many of the characters are underlined, so the best way to approach the statement with the cursor is from the top.

Thus, press the (↖) key so that the cursor moves directly to the "home" position of the display. Then, continue to press the (↓) key until the cursor rests under 1000 in the statement you just created. Then press (END LINE). Now move the cursor with the (↓) key until it rests under 2000 in the second statement you created and press (END LINE).



```
DISP "1000 M$="&CHR$(34)&M$&CHR$
(34)
1000 M$="ᴇ◀├◀?@▒ᴇ‡xñ⊥ñΣΓaτa▒▒├⊢⊢
⊢Sa‡_▒_◀_◀ᵬ¿⊥?x?ᴇ├`├◀ᴇ◀"

DISP "2000 M2$="&CHR$(34)&M2$&CH
R$(34)
2000 M2$="ᴇᴇ◀├├◀?⊢@▒aᴇ‡wxñâ⊥ñzΣΓ
⊥aτaa▒◀▒├◀⊢⊢◀⊢S<a‡P_▒@_◀__◀ᵬᵬ<±⊥
?Gx?0ᴇ├▒`├├◀ᴇᴇ◀"
```

Move the cursor here, then press (END LINE).

Then move the cursor here and press (END LINE).

Since you have now added variable assignment statements for M$ and M2$ to the program, you can delete all of the `READ` and `DATA` statements. Execute:

```
DELETE 10,80
DELETE 105,210
```

Now add the following lines to your program:

```
115 GOSUB 1000
300 STOP
3000 RETURN
```

Go to 1000 to assign values to variables m$ and M2$.
Add `STOP` so that you do not inadvertantly access the subroutine.
Add return statement at end of subroutine.

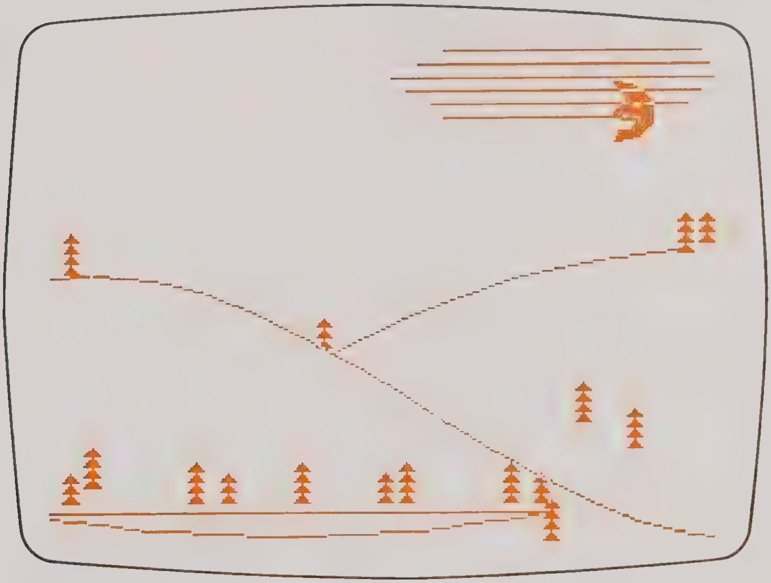
Now list your program; it should match the program listed below:

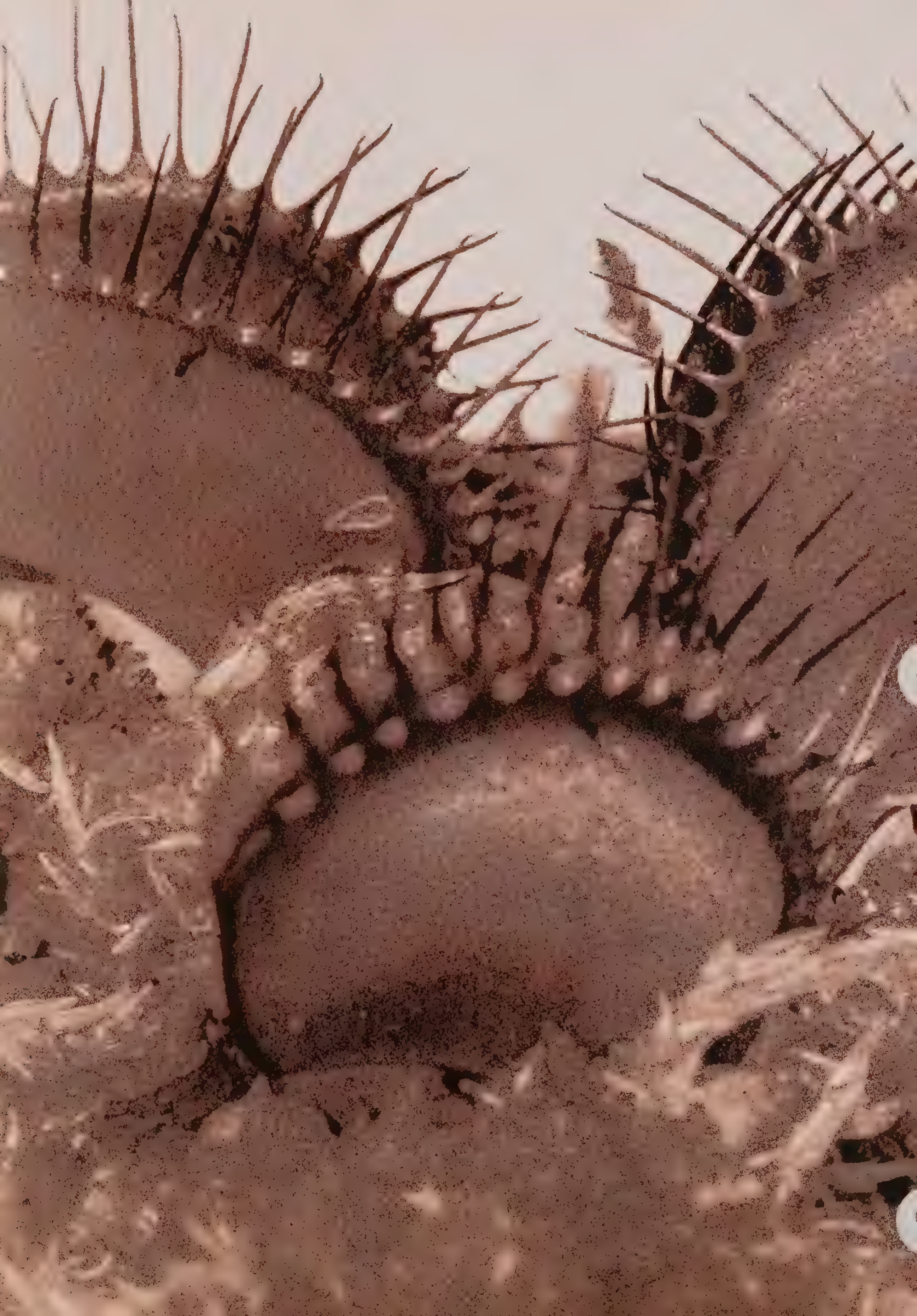
```
  10 REM *BUILD MOON STRING*
  20 DIM M$[46],M2$[69]
  90 SCALE 0,255,0,191
 100 PEN 1 @ GCLEAR
 110 MOVE 0,191
 115 GOSUB 1000
 120 BPLOT M$,2
 210 REM *NOW SET UP LOOPS SO THE
         MOON MOVES FROM LEFT TO
         RIGHT ACROSS THE DISPLAY
 220 REM AND FROM TOP TO BOTTOM
         RIGHT CORNER*
 230 FOR Y=191 to 0 STEP -1
 240 FOR X=0 TO 255 STEP 8
 250 MOVE X,Y
 260 BPLOT M2$,3
 270 WAIT 1000
 280 NEXT X
 290 NEXT Y
 300 STOP
1000 M$="ε◀⊢◀?@▒ε‡xÄ!ÄΣΓατα▒▒⊢⊢⊢
         ⊢Sα‡_▒_◀_◀n¿!?x?ε⊢`⊢◀ε◀"
2000 M2$="εε◀⊢⊢◀?⊢@▒αε‡ωxÄi!ÄxΣΓ
         iατaα▒◀▒⊢◀⊢⊢◀⊢S<α‡P_▒@_◀__◀
         nα¿÷!?Gx?Ωε⊢▒`⊢⊢◀εε◀"
3000 RETURN
9999 END
```

If you run the program once again, the man in the moon will move across the display as it did earlier with the program on pages 249 and 250.

Problem

12.7 Create a scene for the man-in-the-moon using trigonometric curves for mountains, a segment of a circle for a lake, `XAXIS` statements for clouds, and our first triangle `BPLOT` for trees. Or better yet, create your own scene, by creating some new `BPLOT` character strings. Here's our scene; a sample program exists in appendix F.





Section 13

Debugging and Error Recovery

Even the most experienced programmers find errors in their programs. These errors range from mistakes in the original formulas to mistakes in the logical flow of the program. Whenever they occur, they need to be found and corrected, and your HP-85 is designed to make various error-checking processes as easy and convenient as possible.

One of the easiest ways to find out if your program is working properly is to work a test case in which you either know the answer or the answer can easily be determined. In lengthy, complex programs, a wrong test-case answer will seldom pinpoint a mistake.

Tracing Program Execution

A convenient method of debugging logic errors in a program is to trace the order of statement execution and variable assignments in a program. The HP-85 provides three tracing statements; `TRACE`, `TRACE VAR`, and `TRACE ALL`, which includes `TRACE` and `TRACE VAR`.

The trace statements can be programmed or executed manually from the keyboard. The trace statements are independent of each other; thus, one or all of the trace statements can be in effect at any time, but `TRACE ALL` includes `TRACE` and `TRACE VAR`.

Tracing Branches

The `TRACE` statement is used to trace the order of statement execution in all or part of a program.

`TRACE`

If the order of program execution proceeds sequentially from the lowest numbered statement to the next higher numbered statement, nothing is printed. But whenever a branch occurs in a program, both the statement number where the branch occurred and the number of the statement to which it branched are printed in the form:

`TRACE LINE` statement number `TO` statement number

Tracing the Values of Variables

You can trace changes in the values of program variables without an output statement by using the `TRACE VAR` *(trace variable)* statement. Calculator mode variables cannot be traced and any attempt to do so will produce an error. Thus, you should always execute `INIT` or `RUN` prior to executing `TRACE VAR`.

`TRACE VAR` variable list

The variable list can contain simple numeric variables, string variables, and references to entire arrays. You can trace as many variables in a program as you wish. The variable names must be separated by commas in the list.

Simple numeric variables and string variables are specified by name; subscripted (array) variables are specified by a name followed by a set of parentheses. A comma may be included within the parentheses to specify a two-dimensional array if desired for documentation purposes.

For example, suppose your program contains simple numeric variables A and B, arrays C(4) and D(25,3), and string E$. To trace all of the variable assignments in your program, execute:

```
TRACE VAR A,B,C(),D(,),E$
```

Use the variable name followed by `()` to denote an array. The use of the comma for two-dimensional arrays is optional.

Whenever a change occurs in the value of the variable(s) you are tracing, the printed trace output indicates the statement number in which the change occured and:

- The name and new value of a simple numeric variable.
- The name, subscript(s), and new value of a particular array element.
- The name of a string variable.
- The name in the form A() or A(,) and the new value of the first element of the array for statements that operate on complete arrays (e.g., `READ#`).

The form of `TRACE VAR` output is:

`TRACE LINE` statement number variable name [`(`subscripts`)`][`=` value]

New values are given only for simple numeric variables and array elements. Only the statement number of the variable change and the name of the variable that changed are given for strings. When an entire array is traced, only the new value of the first element is given, along with the statement number of the array change and the array name.

Trace All Statements and Variable Assignments

The `TRACE ALL` statement enables you to trace the order of program execution for every statement in a program and the value change of every variable in the program.

```
TRACE ALL
```

The `TRACE ALL` statement is most often used with the STEP key, as we'll see shortly.

Since the tracing operations output all information to the printer, you can save paper by executing the `PRINTER IS 1` statement before executing the program that you are tracing.

Cancelling Trace Operations

All trace operations are cancelled by executing `SCRATCH` or the `NORMAL` statement:

```
NORMAL
```

Example: Load and run the following Base Conversions program to find the octal representation of the binary number 10101010. Execute the `TRACE` statement before you run the program to follow the order of program execution.

```
 10 REM *NUMBER BASE CONVERSION
 20 DIM B$[16],I$[24],C$[24]
 30 B$="0123456789ABCDEF"               Assigns check string.
 40 DISP "INPUT BASE, OUTPUT BAS        Inputs number bases.
    E";
 50 INPUT B1,B2
 60 DISP "NUMBER IN BASE";B1;           Inputs number in first base.
 70 INPUT I$
 80 N=0
 90 FOR I=1 TO LEN(I$)                  Initializes base 10 value.
100 P=POS(B$[1,B1],I$[I,I])             Checks for illegal character.
110 IF P=0 THEN 270                     If illegal character go to 270.
120 N=B1*N+P-1                          Accumulate equivalent in base 10.
130 NEXT I
140 O$=""                               Initializes output string.
150 N=N/B2                              Shifts one digit to the right.
160 P=B2*FP(N)+1                        Gets character.
170 O$=O$&B$[P,P]                       Build string in reverse order.
180 N=INT(N)
190 IF N#0 THEN 150                     Checks if done.
200 PRINT I$;"BASE";B1                  Prints input.
210 FOR I=LEN(O$) TO 1 STEP -1          Prints output string.
220 PRINT O$[I,I];
230 NEXT I
240 PRINT " BASE";B2
250 PRINT
260 STOP
270 BEEP                                Illegal character input.
280 PRINT I$[I,I];"NOT ALLOWED
    IN BASE";B1
290 PRINT
300 GOTO 60
310 END
```

We executed the `PRINT ALL` command to get the listing below:

```
TRACE
RUN
INPUT BASE, OUTPUT BASE?
2,8
NUMBER IN BASE 2?
10101010
 Trace line 130 to 100
 Trace line 130 to 100
 Trace line 130 to 100
 Trace line 130 to 100
 Trace line 130 to 100
 Trace line 130 to 100
 Trace line 130 to 100
 Trace line 190 to 150
 Trace line 190 to 150
10101010 BASE 2
 Trace line 230 to 220
 Trace line 230 to 220
252 BASE 8
```

As you can see, the program executes the same set of branching operations, from 130 to 100, seven times in order to convert the original number to its decimal equivalent.

Then the program executes line 150 through 190 three times (two branches from 190 to 150) to build the character string of its octal equivalent.

Finally, the string is output with the `FOR-NEXT` loop.

The program exits the `FOR-NEXT` loop in statements 90 through 130 when the decimal equivalent of the original value has been accumulated and stored in the variable N. Regardless of the base of the original number, or the base that you convert it to, you can find the decimal equivalent of the original number by using the `TRACE VAR` statement with this program.

Using the `TRACE VAR` statement, trace variable N to find the decimal equivalent of 1321 base 4 in the process of converting it to base 3.

First, execute `NORMAL` to cancel our previous `TRACE` statement.

```
NORMAL
PRINT ALL
TRACE VAR N
RUN
INPUT BASE, OUTPUT BASE?
4,3
NUMBER IN BASE 4?
1321
 Trace line 80 N=0
 Trace line 120 N=1
 Trace line 120 N=7
 Trace line 120 N=30
 Trace line 120 N=121
 Trace line 150 N=40.3333333333
 Trace line 180 N=40
 Trace line 150 N=13.3333333333
 Trace line 180 N=13
 Trace line 150 N=4.33333333333
 Trace line 180 N=4
 Trace line 150 N=1.33333333333
 Trace line 180 N=1
 Trace line 150 N=.333333333333
 Trace line 180 N=0
1321 BASE 4
11111 Base 3
```

`NORMAL` also cancels `PRINT ALL`.
Resets print all mode.
Traces variable N.

Decimal equivalent of 1321_4;
The largest value of N.

You can use any of the `TRACE` statements in a program. For instance, add the following statements to the Base Conversions program.

```
85 TRACE VAR O$,B$,N
135 NORMAL @ PRINT ALL
```

Now execute `NORMAL` to cancel our previous `TRACE VAR` statement and run the program to find the binary equivalent of 86_9.

```
NORMAL
PRINTALL
RUN
INPUT BASE, CUTPUT BASE?
9,2
NUMBER IN BASE 9 ?
86
 Trace line 120 N=8
 Trace line 120 N=78
86 BASE
1001110 BASE 2
```

Since the values of variables O$ and B$ do not change between statements 85 and 135, *no* `TRACE VAR` output occurs for them.

The STEP Key

You can execute a program one line at a time by using the (STEP) key. If a program has been halted by a `PAUSE` statement or the (PAUSE) key, it can be continued, one line at a time, by pressing the (STEP) key. As soon as (STEP) is pressed, the line designated by the internal program counter is executed, then the program halts again.

You can execute an entire program—one step at a time—by first initializing the program with the (INIT) key (or by executing `INIT`), and then by pressing (STEP) to execute each program statement.

Since the (STEP) key does not output anything to the printer or display, it is often desirable to execute the `TRACE ALL` statement so that you know what line is being executed.

Example: Execute the `TRACE ALL` statement and then execute the Base Conversions program, one statement at a time using the (STEP/PAUSE) key. (Remember to delete statement 135, from out last example, so that the tracing operations are not cancelled.) First initialize the program with the `INIT` command.

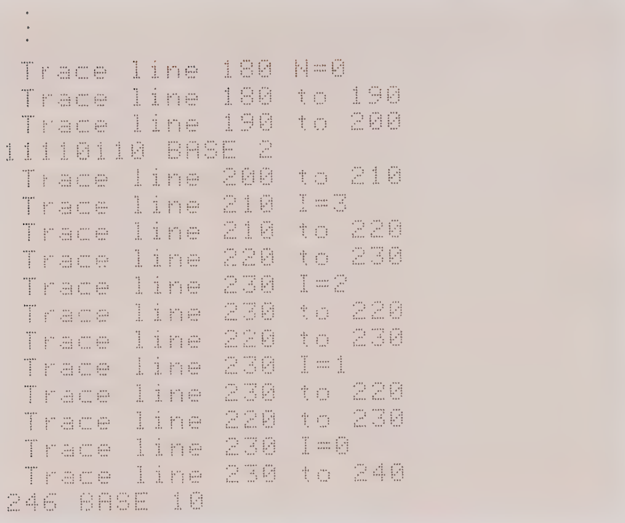
```
 PRINT ALL
 INIT
 TRACE ALL

 Trace line 10 to 20
 Trace line 20 to 30
 Trace line 30 B$
 Trace line 30 to 40
 Trace line 40 to 50
INPUT BASE, OUTPUT BASE?
2,10
 Trace line 50 B1=2
 Trace line 50 B2=10
 Trace line 50 to 60
 Trace line 60 to 70
NUMBER IN BASE 2 ?
11110110
 Trace line 70 I$
 Trace line 70 to 80
 Trace line 80 N=0
 Trace line 80 to 90
 Trace line 90 I=1
 Trace line 90 to 100
 Trace line 100 P=2
 Trace line 100 to 110
 Trace line 110 to 120
 Trace line 120 N=1
 Trace line 120 to 130
 Trace line 130 I=2
 Trace line 130 to 100
 Trace line 100 P=4
 Trace line 100 to 110
 Trace line 110 to 120
 Trace line 120 N=[illegible]
 Trace line 120 to 130
 Trace line 130 I=3
 Trace line 130 to 100
 .
 .
 .
```

Initializes the program.

`TRACE ALL` traces all variables and branches.

Now press the (STEP) key to execute the program one statement at a time.

Be sure to press (END LINE) to enter data requested by an `INPUT` statement. Then continue stepping through the program by pressing the (STEP) key. If you press the (STEP) key in response to an input statement, the keycode for (STEP), *θ*, will appear. Simply backspace and enter the data with (END LINE).



```
  .
  .
 Trace line 180 N=0
 Trace line 180 to 190
 Trace line 190 to 200
11110110 BASE 2
 Trace line 200 to 210
 Trace line 210 I=3
 Trace line 210 to 220
 Trace line 220 to 230
 Trace line 230 I=2
 Trace line 230 to 220
 Trace line 220 to 230
 Trace line 230 I=1
 Trace line 230 to 220
 Trace line 220 to 230
 Trace line 230 I=0
 Trace line 230 to 240
246 BASE 10
```

Checking a Halted Program

Various operations that aid in debugging can be performed on a program halted by PAUSE or an `Error` or before pressing STEP or CONT:

- Values of variables can be checked merely by keying in the variable names followed by END LINE.
- Values of variables can be assigned or changed if statements like `A(5,2)=7` or `B=0` are executed.
- Most program statements can be executed without statement numbers, like `PRINT`, `DISP` or `COPY`.
- Arithmetic operations can be performed and math functions can be executed.
- Any system command can be executed—`PRINT ALL`, `DEG`, `GRAD`, or `RAD`, etc.

If the halted program is continued with either the CONT or STEP key, any of the previously mentioned operations that affect program execution remain intact. For instance, values of variables that were changed retain their new values; a `DEG` statement causes the program to calculate angles in degrees, etc.

If, however, the halted program is restarted with a `RUN` command, or `INIT` is executed before you continue, then the program is initialized so that all variables start with undefined values until reassigned a value in the program (or from the keyboard). (Refer to appendix C to see what is affected by `RUN` and `INIT`.) You cannot continue a program with STEP or CONT after editing a line of the program; the program must first be initialized with `RUN` or `INIT`.

Error Testing and Recovery

Run time errors are those that occur only when a program is running. Division by zero is an example. These errors normally halt execution of a running program. The `DEFAULT ON` statement (which is set at power on) provides default values for some error-causing operations, displays a warning message, and thus allows your program to continue. There is more than one way to control program errors or catch a bug. Through use of the `ON ERROR` statement, routine errors can be recovered so that execution can continue with the specified line after execution of the line in which the error occured.

The ON ERROR statement specifies a branching that takes place after an error occurs.

ON ERROR GOTO statement number
ON ERROR GOSUB statement number

The ON ERROR statement declares what should happen if an error occurs. It need be executed only once in each program segment to establish the ON ERROR condition. Execution of another ON ERROR statement replaces the previous one.

When a run-time error occurs and the ON ERROR condition has been established, execution is transferred to the specified line. Then the ERRN and ERRL functions (discussed next) could be tested and error recovery procedures could be executed. The error is "ignored" if the statement referenced by a GOSUB is a RETURN statement. Execution continues with the statement after the one in which the error occured.

If the recovery routine also contains an error, it is possible to program into an endless loop. The program can be halted, of course, by pressing RESET.

CAUTION

It is not recommended to ignore service errors. Error numbers 65, 73, and 74 can signify a defective cartridge or defective tape transport. Refer to appendix B.

An ON ERROR statement is disabled with the OFF ERROR statement:

OFF ERROR

It is usually a good idea to turn off the ON ERROR condition as soon as an error has been found so that you don't trap out unexpected errors.

Two numeric functions are associated with ON ERROR:

ERRL The *error line* function outputs the line number in which the most recent program execution error occured.

ERRN The *error number* function outputs the number of the most recent program execution error. Appendix E contains a complete list of error numbers and messages.

Example: Suppose you are concerned that a certain computation will cause a numeric overflow. If it does, you want to skip that segment of the program and go to another part of the program. If the error is not a numeric overflow error, you want to display the number of the statement in which the error occurred and pause so that you can do some program checks. Write the necessary statements that would carry out these functions.

In the following program, we create some obvious errors so that the order of program execution may be followed easily.

```
 10 N=1.E400
 20 DISP N
 30 M=1.E400
 40 DISP M
 50 ON ERROR GOTO 100
 60 K=N*M
 70 OFF ERROR
 80 DISP K
 90 GOTO 510
100 OFF ERROR
110 IF ERRN=2 THEN 500

120 DISP "ERRN=";ERRN;"ERRL=";E
    RRL
130 PAUSE
500 DISP "ARRIVED AT STATEMENT
    NUMBER 500"
510 END
```

If an error occurs, go to 100.

We turn off the error overriding condition as soon as the error occurs, or the expected error-causing statement is executed so that we don't trap out errors we're not prepared to handle.

If error number 2 (overflow), then go to 500.

If not an overflow, display error number and error line.

Then pause.

If an overflow condition occurs, will skip to this statement.

Now run the program:

```
RUN
 1.E400
 1.E400
ARRIVED AT STATEMENT  NUMBER 500
```

As you can see, the program checked for an overflow condition, then skipped to statement 500.

Now change statement 60 to read:

```
60 K=N/0
```

Creates a division by zero error.

And run the program again:

```
RUN
 1.E400
 1.E400
ERRN= 8 ERRL= 60
```

Displays error number for division by zero and the line number in which the error occured, then pauses.

Since the program has paused, you may perform various program checks, or change the values of variables before you continue.

Some Hints About the System

Occasionally, your program may not work the way you think it should, perhaps by giving erroneous results, and yet the system does not detect any errors. The following list of things to remember about the HP-85 system may help you in detecting program errors or user-misunderstandings.

- Be sure to close data files if the program halts because of an error in the midst of printing to a file. Remember that the system uses buffers to ''write'' data to the tape and that the buffers are ''dumped'' only under specific conditions (refer to page 184).

- If an assignment (e.g., J=5) is made before a program is initialized with RUN or INIT, it is a calculator mode variable, not known to the program. However, when an assignment is made after a program is allocated (initialized) to a variable that is known to the program, then the assignment is made to that program variable. If the variable is not known to the program (i.e., not referenced in the program), then it is a calculator mode variable.

- Error messages report the first error that occured; there may be others. Remember that the system tries to interpret an expression as a statement and then as an expression. If you get an error with a calculator mode expression, try executing the same expression in a DISP statement. Then the system will know that it is looking at an expression in a DISP statement and you'll get a better error message. A bad expression is considered a bad statement if typed as [expression] (END LINE).

- The three programmable timers are extremely useful, but be aware that they interrupt the system at the frequency you specify until they are turned off by executing (SCRATCH), (RESET), or OFF TIMER#. Thus, for small interrupt intervals, timers can have an adverse effect on the execution speed of the system.

- The XAXIS and YAXIS statements are interruptable during tic-generation by pressing (RESET)—just in case you specify very small tic spacing that might take hours to complete.

- Programs are stored allocated (including the space required for dedicated variables) unless they contain variables in common (with a COM statement) or have allocation errors (e.g., missing a reference line). If your program contains dimensioned variables, you may want to add a token COM statement (e.g., 1 COM Z9) in order to deallocate the program before it is stored.

- If you reference a multiple-line function in a PRINT or DISP list, and the function contains a PRINT or DISP statement, you may not get the output you expect.

- Should you get a memory overflow error when attempting to read a long string from a data file, break the string into shorter substrings and write the substrings into smaller logical records. Then read the substrings back, one at a time, into computer memory.

Memory Conservation Hints

- Remarks and comments (!) take one byte per character. Use enough to document your program but don't be excessively wordy.

- Use INTEGER and SHORT data types for arrays whenever possible.

- Use INTEGER constants instead of REAL constants whenever possible (e.g., 4 instead of 4.).

- Explicitly dimension *all* arrays if the upper bound is not 10.

- Explicitly dimension all strings if the maximum length is 10 or less.

- Use OPTION BASE 1 if you don't plan to use the zero'th element of your arrays.

- Use multistatement lines (using @) when it doesn't detract from program readability and if the program is not intended to be run on other BASIC computers.
- Use a variable assignment for program constants that occur more than once. Variable names take up less space.
- Use subroutines or functions for program sequences that occur more than once.
- Try to reuse variables when possible, rather than declaring new variables.

Appendix A

Accessories

Standard Accessories

Your HP-85 comes equipped with one each of the following standard accessories:

Accessory	Part Number
• *HP-85 Owner's Manual and Programming Guide*	00085-90002
• *HP-85 Pocket Guide*	00085-90040
• *HP-85 BASIC Reference Card*	00085-90039
• Standard Pac, including: Instruction Manual Preprogrammed Tape Cartridge	00085-13001
• Registration Card	
• Service Card	
• Accessory Data Sheet	
• Users' Library Form	
• Roll of Thermal Printer Paper	
• Power Cord	
• Fuses and Fuse Cap Holders 750 milliamperes fuse (for 115 Vac-nominal line voltage) and U.S. style fuse cap holder T400 milliamperes fuse (for 230 Vac-nominal line voltage) and European style fuse cap holder	
• Three-Ring Binder and Dividers	

Optional Accessories

In addition to the standard accessories shipped with your HP-85, Hewlett-Packard also makes available the following optional accessories. These have been created to help you maximize the usability and convenience of your personal computer.

HP-85 Applications Pacs

Each pac offers one or more BASIC programs in a particular field or discipline prerecorded on a tape cartridge. Each pac comes complete with a detailed instruction manual and handy pac binder that carries up to four cartridges and the instruction manual.

HP-85 Plug-in Modules

- HP 82903A 16K RAM (Random Access Memory)
  The 16K memory module approximately doubles the amount of space for programming and data storage.
- Enhancement ROMs (Read Only Memory)
  A ROM drawer and several plug-in ROMs will be released in the near future that actually enhance the capabilities of your HP-85. For more information about HP-85 enhancement ROMs call the following toll-free number: 800/547-3400 (except in Alaska and Hawaii).

Owner's Manual and Pocket Guides

Supply members of your staff with personal copies of HP-85 operating information:

• *HP-85 Owner's Manual and Programming Guide*	00085-90002
• *HP-85 Pocket Guide*	00085-90040
• *HP-85 BASIC Reference Card*	00085-90039

HP-85 Carrying Case (HP 82933A)

With its stylish, leather-like exterior, made of durable, easy-to-clean vinyl, the lightweight HP-85 carrying case provides you with a convenient means of transporting your computer safely. Inside the case, molded foam liners conform exactly to the contours of the HP-85, providing maximum shock absorption. Designed to carry the computer, enhancement plug-in modules, and power cord, the carrying case also has an exterior-accessible pouch that will hold two instruction manuals, a roll of paper, and several tape cartridges. The case is secured with a three-sided zipper and is fitted with a double-web handle with keeper, providing a suitcase-type grip. Dimensions:

23 centimeters (9 inches) thick
48 centimeters (19 inches) wide
56 centimeters (22 inches) high

BLANK TAPE CARTRIDGES (HP 98200A)

Hewlett-Packard blank tape cartridges are available in packages of five each.

Tape Cartridge Binder (HP 82932A)

The tape cartridge binder provides you with a convenient way to both store and transport your HP-85 tape cartridges and one instruction book. Available in vinyl, the case measures 29 cm (11.5 in) high, 28 cm (11 in) wide, and 5 cm (2 in) deep, and has space for four Hewlett-Packard tape cartridges.

Thermal Printer Paper (HP 82931A)

Each pack gives you two rolls of special HP-85 thermal printer paper. Roll length: 120 meters (400 feet).

Three-Ring Manual Binder and Dividers (HP 82935A)

Additional HP-85 manual binders are available, enabling you and members of your staff to organize your HP-85 system user's manuals conveniently. The binder, measuring 29 cm (11.5 in) high, 28 cm (11 in) long, and 6.5 cm (2.5 in) wide, includes sheet lifters and a set of dividers.

Appendix B

Owner's Information

The following information covers the initial set-up of your HP-85 Personal Computer and includes other information that is important when you first receive the computer.

> **Note:** Become thoroughly familiar with the information in this appendix before attempting to operate your HP-85.

Inspection Procedure

Your HP-85 is another example of the award-winning design, superior quality, and attention to detail in engineering and construction that have marked Hewlett-Packard electronic instruments for more than 30 years. Each Hewlett-Packard computer is precision crafted by people who are dedicated to giving you the best possible product at an affordable price.

Your HP-85 computer was thoroughly inspected before shipping and should be ready to operate after completing the set-up instructions. Carefully check the computer for any physical damage sustained during shipment. Do not turn the power on if the CRT display shows any cracks. Notify your dealer and file a claim with any carriers involved if there is any such damage.

Please check to ensure that you have received all of the standard accessories included with the HP-85. Review the list of standard accessories in appendix A. If any accessory items are missing, please contact the dealer from whom you purchased the computer. If your computer was purchased directly from Hewlett-Packard, please contact the office through which your order was placed.

Power Supply Information

Power Cords

Power cords supplied by HP have polarities matched to the power-input socket on the machine, as shown below.

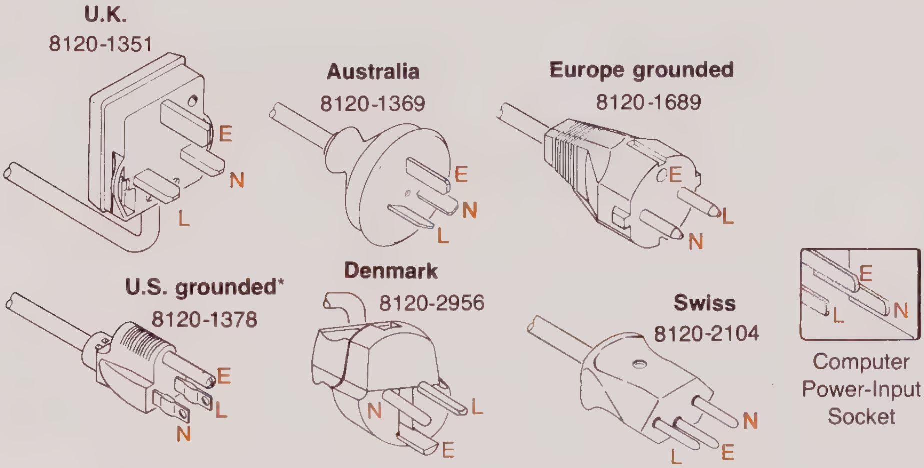
- L=Line or Active Conductor (also called "live" or "hot")
- N=Neutral or Identified Conductor
- E=Earth ground

WARNING

Use only the HP-85 power cord specified by Hewlett-Packard for your area.

If it is necessary to replace the power cord, the replacement cord must have the same polarity as the original. Otherwise a safety hazard from electrical shock to personnel might exist. In addition, the equipment could be extensively damaged.

Power cords with different plugs are available for the HP-85; the part number of each cord is shown below. Each plug has a ground connector. The cord packaged with the machine depends upon where the machine was delivered. If your equipment has the wrong power cord for your area, please contact your local authorized HP-85 dealer or HP sales and service office for information on how to obtain the proper cord.


U.K.
8120-1351
E
N
L
Australia
8120-1369
E
N
L
Europe grounded
8120-1689
E
L
N
U.S. grounded*
8120-1378
E
L
N
Denmark
8120-2956
N
L
E
Swiss
8120-2104
N
L
E
E
L
N
Computer Power-Input Socket

Grounding Requirements

To protect operating personnel, the National Electrical Manufactures' Association (NEMA) recommends that all equipment not double insulated be properly grounded. The HP-85 is equipped with a three-conductor power cable which, when connected to an appropriate power receptacle, grounds the machine. To preserve this protection feature, do not operate the machine from a power outlet which has no earth ground connection.

WARNING

To avoid the possibility of any injury, disconnect the ac power cord before installing or replacing a fuse.

Power Requirements

The HP-85 has the following power requirements:

Line Voltage	
115 Vac Nominal	100/117 Vac
230 Vac Nominal	220/240 Vac
Line Frequency	50/60 Hz
Power Consumption	40 Watts Nominal

Fuses

For 100/117 Vac operation, set the voltage selector switch to 115V and use a 750mA fuse; for 220/240 Vac operation, set the voltage selector switch to 230V and use a T400mA fuse.

---

* UL and CSA approved for use in the U.S. and Canada with machines set for 115 Vac operation.

Service

WARNING

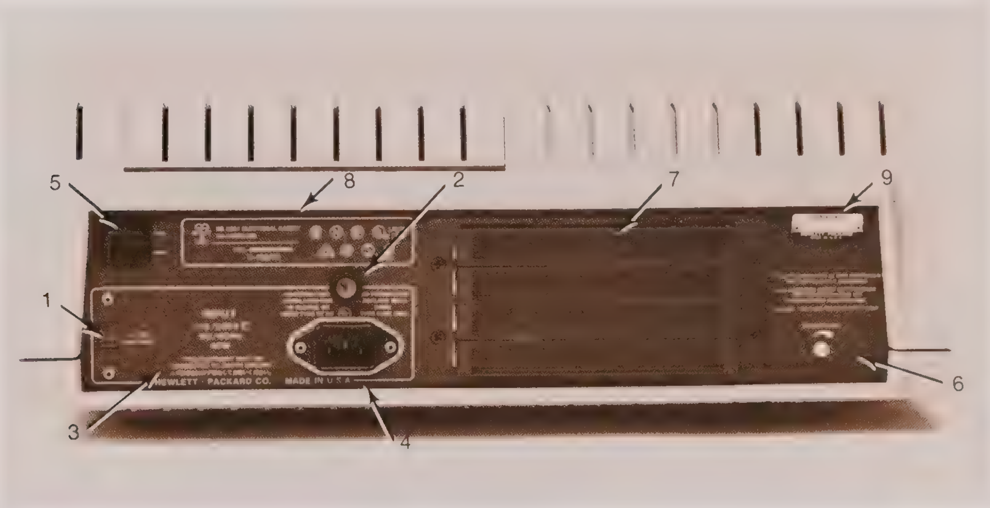
High voltages are present inside the HP-85. There are no customer serviceable parts inside the HP-85. In case of any difficulty or malfunction with your HP-85 contact your nearest authorized HP-85 dealer or HP repair facility.

For specific warranty and service information, refer to pages 287 through 289.

Rear Panel

Understanding the rear panel layout and features of your HP-85 computer is important for safe and efficient operation. The rear panel contains the following:

1. Line Voltage Selector Switch.
2. Fuse Receptacle.
3. Ground Information.
4. Power Cord Receptacle.
5. ON-OFF Switch.
6. Display Brightness Control.
7. Module Plug-in Ports with Covers.
8. Worldwide Saftey Approval Nomenclature.
9. Serial Number Plate.



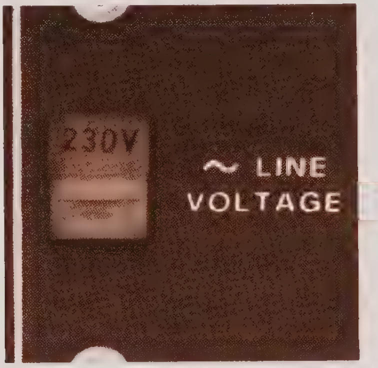
Initial Set-up Instructions

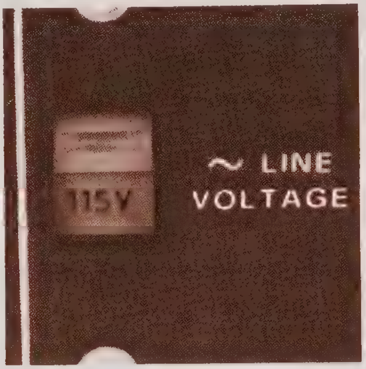
1. Disconnect the power cord and make sure the ON/OFF switch is OFF.
2. Ensure that the voltage selector switch located on the rear panel of the computer is set for the voltage range of the nominal line voltage in your area.

CAUTION

Check the selector switch before applying power. Damage to the computer will occur if the selector switch is set to 115 volts ac, and 230 volts ac is applied to the power input connector.

If it is necessary to alter the setting of the switch, insert the tip of a small screwdriver or coin into the slot on the switch. Slide the switch so that the position of the slot corresponds to the desired voltage as shown below. The computer is shipped with the voltage selector in the 230 Vac position.



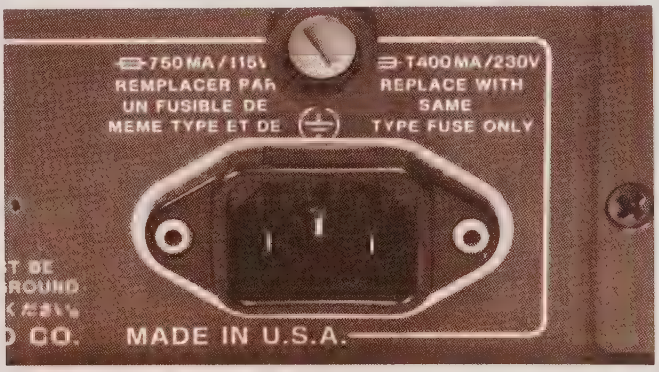


WARNING

Before installing or replacing a fuse, be sure that the computer is disconnected from any ac power source. Otherwise, a chance of electrical shock to personnel exists and the new fuse might be immediately overloaded.

3. Next install the proper fuse.

   The computer's fuse receptacle is located on the rear panel. (See figure below.) A 750 mA fuse is required for 115 Vac operation and a T400 mA fuse is required for Vac operation.



   The photograph shows the location of the fuse receptacle on the rear panel. To install or replace the fuse, first disconnect the power cord from the machine. Install or replace the proper fuse in the fuse cap holder (either end of the fuse can be inserted into the cap). Now, install the fuse and fuse cap into the fuse receptacle by pressing the cap inward and at the same time turning it clockwise until it locks in place.

4. Now, connect the power cord to the power input receptacle on the back of the computer. Plug the other end of the cord into the ac power outlet.

5. Switch the HP-85 on using the switch on the upper left side of the rear panel. A cursor (underscore) should appear in the upper left corner of the CRT display within 2 to 3 seconds. Each time the power is turned on, the system performs a self-test operation. When the cursor appears on the screen, the HP-85 is ready to go to work.

The brightness of the display can be adjusted using the Brightness knob on the lower right side of the rear panel.

Should the cursor not appear or the words `Error 23 : SELF-TEST` appear on the display, turn the machine off, then on again. Should the problem persist, contact your local authorized HP-85 dealer or HP sales and service office.

Installing Plug-In Modules

Your HP-85 is designed with four module ports on the rear panel. The ports are numbered 1 through 4 from the top. Before shipping from the factory, each port is fitted with a removable protective cover. It is recommended that each port be kept covered when not in use.

First we will discuss general module installation and removal, then we will discuss the installation of plug-in ROMs into a special ROM drawer module.

WARNING

Do not place fingers, tools, or other foreign objects into the plug-in ports. Such actions may result in minor electrical shock hazard and interference with pacemaker devices worn by some persons. Damage to plug-in port contacts and the computer's internal circuitry may also result.

General Module Installation and Removal

The HP-85 plug-in modules may be installed or removed as often as your needs require. To install modules, observe the following procedures.

1. Read all documentation accompanying each module for user instructions, warnings, and any limitations.

CAUTION

Always switch off the machine and any peripherals involved when inserting or removing modules. Use only the plug-in modules designed by Hewlett-Packard specifically for the HP-85. Failure to do either could damage equipment.

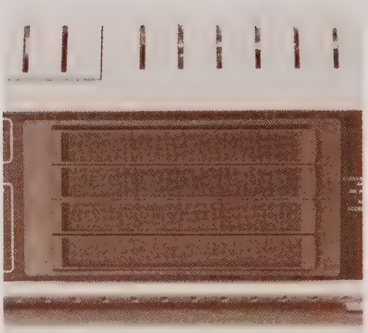
2. Turn off your HP-85 system. If an interface module is to be installed, or is already in use, switch off any peripheral devices involved.

CAUTION

If a module jams when inserted into a port, it may be upside down or designed for another port. Attempting to force it further may result in damage to the computer or the module.

To insert a plug-in module:

1. Remove the protective cover from the plug-in port to be used.



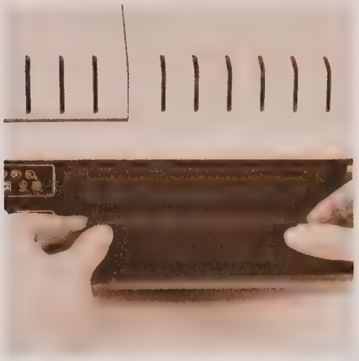
Note: Most plug-in modules can be inserted in any of the four ports. However, examine the documentation included with each module for any instructions regarding the use of a specific plug-in port. If it is intended that a module fit into a particular port, it can be inserted only in that port.

2. With the label right-side-up, insert the contact end of the module into the port and push until the module seats firmly with its stops against the port's edge. A slight up and down motion may be necessary to start the module moving in the tracks of the port. The tracks are keyed to prevent upside-down module insertion.

To remove a plug-in module, observe the following procedure:

1. Switch OFF your HP-85 system and any connected peripherals.
2. Firmly grasp and pull the module free of the port. Store the module in its original container or where it will be safe from damage to the contacts.

3. Replace the port cover.



Note: Up to four different modules can be installed in the HP-85 at any time. However, do not install duplicate RAM modules or duplicate ROMs into the ROM drawer. Such duplication can create error conditions and will not increase memory or computing power.

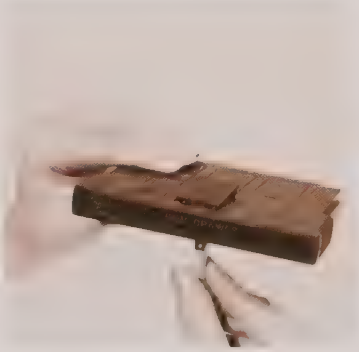
Plug-in ROM Installation and Removal

The ROM drawer is a particular plug-in module that contains six rectangular slots for individual plug-in ROMs, each fitted with its own protective cap.

Any HP-85 plug-in ROM will fit in any of the six positions in the ROM drawer. Be sure to read all documentation accompanying each plug-in ROM for user instructions, warnings, and any limitations. Remember that duplicate ROMs will *not* increase your computing power and may even create error conditions.

To insert a plug-in ROM into the ROM drawer:

1. Remove the protective cap from the desired plug-in slot in the ROM drawer as follows:
   - Insert the eraser end of a pencil into the circular hole on the underside of ROM drawer.
   - Press with the pencil until the cap snaps off.



CAUTION

Do not touch the spring-finger connectors in the ROM drawer with your fingers or insert tools or other foreign objects. Static discharge could damage electrical components.

2. Inside each plug-in slot in the ROM drawer you can see two rows of spring-finger connectors. These connectors correspond to the two rows of holes on the underside of the ROM plug. ROMs can be inserted in only one direction. Insert the ROM plug into the slot with its label up and its beveled edge toward the connector side of the ROM drawer. Push the ROM into place so that the top of the plug is flush with the top of the ROM drawer.

Note: Leave the cap on any slot in the ROM drawer that is not in use.

3. When all of the desired plug-in ROMs have been inserted into the ROM drawer, the module may be installed into a plug-in port on the rear panel of the HP-85 as described under General Module Installation and Removal.

To remove a plug-in ROM from the ROM drawer:

1. First remove the ROM drawer as described under General Module Installation and Removal.
2. Insert the eraser end of a pencil into the hole on the underside of the ROM drawer corresponding to the ROM you wish to remove, just as you did to remove the protective cap. Push gently with the pencil until the ROM pops out.

3. Replace the protective cap over the slot in the ROM drawer.

Your HP-85 Printer

The printer in your HP-85 is a thermal printer that uses a moving print head to print on a special heat-sensitive paper. When the print head is energized, it heats the paper beneath it. The heat causes a chemical reaction in the paper, which then changes color. The printer, designed expressly for the HP-85, prints quickly and quietly at 2.6 lines per centimeter (6.7 lines per inch) at about two lines per second.

Graphics output is uni-directional and, therefore, approximately half the normal print speed.

Paper for Your HP-85

Because the printer in your HP-85 is a thermal printer, it requires special heat-sensitive paper. You should use only the Hewlett-Packard thermal paper available in 400-foot long rolls from your nearest authorized HP-85 dealer or HP sales and service center, or in the U.S., by mail from:

Hewlett-Packard
Corvallis Division
P.O. Box 999
Corvallis, Oregon 97330

Because of the special heat-sensitive requirements of the paper, impact printer paper will *not* work in the HP-85. Also, since different types of thermal paper vary in their sensitivities and abrasiveness, the use of thermal paper other than that available from Hewlett-Packard may result in poor print quality and excessive print head wear.

CAUTION

Use only Hewlett-Packard paper in your HP-85 computer. Failure to do so may result in excessive print head wear.

The heat-sensitive paper used in your HP-85 should be stored in a cool, dark place. Discoloration of paper may occur if it is exposed to direct sunlight for long periods of time, if storage temperatures rise above 65°C (149°F), if the paper is exposed to excessive humidity or to acetone, ammonia, alcohols, or other organic compounds, or if you attempt to erase anything on the paper. (Exposure to gasoline or oil fumes will not harm your HP-85 paper supply.)

Printed paper from your HP-85 will last 30 days or more without fading under fluorescent light, but to ensure the permanence of your records, you should store printed paper at room temperature in a dark place away from direct sunlight, heat, or fumes from organic compounds.

Loading Printer Paper

Printed paper is loaded by using the following procedure. To perform the following steps, the computer must be switched ON.

1. Open the hinged access cover by gently lifting the front edge of the cover up and back until it stops.

2. Remove the empty paper core with the roll guides from the paper well by pulling gently until the roll guides are released from their sockets. Discard the old paper core but save the roll guides at either end of the paper core. Remove any paper remaining from the previous roll by pressing the PAPER ADV key until the remaining paper stops moving. Then lift the paper out of the printer mechanism.

3. Discard the first 1-1/2 turns of the new roll to insure that no glue, tape, or other foreign matter is on the paper. Make sure that the leading edge of the paper is straight and cleanly cut or folded. A crooked or jagged leading edge will not engage properly in the paper advance rollers.

4. Insert the cylindrical ends of the roll guides into the core of the paper roll, aligning the tabs of the roll guides vertically. Using both hands to hold the roll guides in place, rest the paper roll on the paper well. Make sure that the leading edge of the paper is positioned to unroll forward from the bottom. Press inward on the roll guide tabs while pushing down on the paper roll, until the guides snap into place.

5. Pull approximately 6 inches of paper out of the roll and evenly insert the leading edge over and into the grey throat of the paper feed. Continue manually feeding the paper until it halts. Press and hold the paper advance key until the leading edge of the paper passes the top edge of the clear plastic tear bar. Close the hinged access cover, keeping the paper clear.

If the paper feeds properly through the printer mechanism but no printing appears on the tape when the printer is operated, the paper roll is probably inserted backwards. The paper is chemically treated and will print on one side only.

Printer Maintenance

The printer in your HP-85, like the rest of the computer, is crafted for engineering excellence and is designed to give trouble-free operation with a minimum of maintenance. All moving parts in the printer mechanism have self-lubricating qualities. No lubrication, cleaning, or servicing of the mechanism is ever required. Setting the printer intensity dial to 5 or more for long periods of time may affect the long-term performance of the printhead. You can extend the life of the printer by setting the printer intensity dial to 4 or less.

CAUTION

You should never attempt to insert a tool, such as a screwdriver, knifeblade, pencil, or other foreign object into the printer or its mechanism. Such actions can damage the platen, as well as other parts of the printer mechanism and will void your warranty.

If the printer paper should become jammed and fail to feed properly, first tear the jammed paper loose from the rest of the roll, then clear it by grasping the leading edge of the paper and pulling it *forward* through the printer mechanism while holding down the PAPER ADV key. Discard any lengths of paper damaged by tears or creases. Then reload as described above.

If the printer paper should become jammed with the leading edge below the tear bar, remove the clear plastic tear bar to reach the leading edge. Tear the jammed paper loose from the rest of the roll and pull it forward through the printer mechanism.

Before you replace the tear bar, reload the paper. Hold the paper back against the platen to ensure that the platen face will be properly located behind the tear bar. Then slide the tear bar back into place.

The Tape Cartridge

The tape cartridge used with the HP-85 computer is a high-quality digital storage medium. This section covers use, specifications, and care of tape cartridges.

Rewinding the Tape

The REW key or `REWIND` statement rewinds the tape to its beginning. Press SHIFT REW or type `REWIND` END LINE to rewind the tape.

General Information for Use

Rewind time	29 seconds
Initialization time	15 seconds
Search speed	60 inches per second
Read/write speed	10 inches per second
Tape length	43 meters (140 feet)
Number of tracks	2 independent tracks
Typical tape capacity	780 program records (195K bytes)
	850 data records (210K bytes)
Tape directory capacity	42 files (directory entries)
Typical access rate (search speed)	7,800 bytes/second
Typical transfer rate	650 bytes/second
Typical tape life (continuous use)	50 to 100 hours
Typical error rate*	<1 in 10^8 bits (that's less than one in every 100 million!)

Inserting the Tape Cartridge

Insert the tape cartridge so that its label is up and the open edge is toward the computer.

The tape drive door opens when the cartridge is pressed against it; the cartridge can then be inserted.

HP-85
REW
LOAD
TEST
STORE
ROLL
DEL
RESET
INIT

---

* This is dependent on the cleanliness of the tape head, tape care, and the cleanliness of the environment.

Removing the Tape Cartridge

The cartridge may be removed by pressing the bar below the tape drive. The tape drive will partially eject the cartridge so that you can remove it freely the rest of the way.

CAUTION

Do not attempt to remove the cartridge while the tape is in motion. Damage to the tape may result.

HP-85
REW
LOAD
TEST
STORE

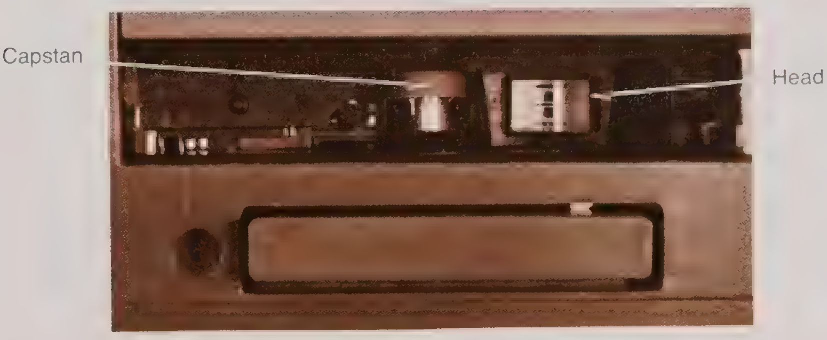
Write Protection

You may protect your cartridge against write (`STORE` or `PRINT#`) operations by sliding the RECORD slide tab to the left before inserting the cartridge. To record on the cartridge, the tab must be all the way to the right.

Tape Care

The cartridge tape drive may develop a buildup of oxide on the recording head after extensive use. As dirty tape drives are one of the most common cause of cartridge-related errors, the following basic precautions are aimed at reducing the risk of cartridge problems in your HP-85.

- Clean the tape head and the tape drive capstan at least as often as every 8 hours of cumulative tape use, or more frequently in dirty environments. Use a cotton-tipped swab dampened with isopropyl alcohol, wiping the tape head and the capstan with a light lateral (back-and-forth) motion (not a heavy scrubbing or up-and-down motion).



The head is the shiny surface on the right rear of the drive.

After using the head cleaning solution, wipe the tape head clean of any residue or lint with a dry cotton swab using a lateral motion (not an up-and-down motion). Be sure the head is dry before inserting a cartridge in the drive. It is a good idea to clean the head before making an important recording.

- Remove the tape cartridge when you are not using the computer. If a cartridge is left in, a flat spot may develop on the rubber wheel of the tape drive capstan in the tape drive of your HP-85. This condition will cause errors when using the tape. The dent is only temporary, and may be corrected by "conditioning" the tape, as described below.

As a normal operating guideline, it is a good practice to run tapes through a conditioning process after every 6 to 8 hours of use. "Conditioning" a tape means to run the tape forward to the end of the tape, reverse it, and run the tape backward to the beginning of the tape. This is done by inserting the tape cartridge to be conditioned, and executing the CTAPE *(condition tape)* command. CTAPE will not affect any programs or data on the tape.

CTAPE (END LINE)

Conditioning is necessary for smooth, continuous operation of the cartridge. (By warming up the tape drive capstan, conditioning also helps to remove a dent caused by leaving a cartridge in the drive.) Whenever a cartridge has been subjected to sudden environmental changes (such as being transported by air), you should condition the tape before use. Also if a READ error occurs while reading a particular cartridge, it may be due to uneven tension on the tape. Conditioning restores proper tension, and the tape will operate smoothly. If READ errors still occur after conditioning, try cleaning the tape head as described above.

- Keep the cartridge in the plastic container supplied with it.
- **Never** eject the tape cartridge while it is moving. Damage to information can be severe if a write or directory operation is in progress.

CAUTION

Strong magnetic fields can erase programs and data stored on tape. Where conditions warrant, keeping cartridges in a metal box, such as a card index, will help protect tapes from potential sources of magnetic damage.

Physical damage to tapes, such as wrinkles or folds, can cause recording and reading problems.

Tape Life

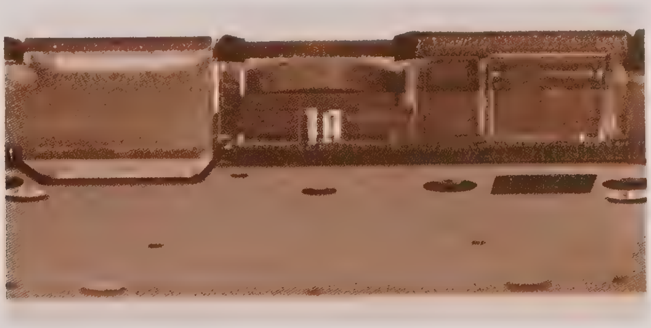
The tape cartridge has a typical life span of 50 to 100 hours of cumulative use. Environmental conditions of 25°C (77°F) and 20 to 50% relative humidity are most favorable for long tape life. A high duty cycle (percent of time the tape is accessed during the total time the computer is in use), high turning resistance, and continuous use for long periods of time (1/2 to 3 hours) contribute to heat buildup in cartridges and decrease tape life. Because tape cartridges eventually wear out, it is always a good practice to maintain back-up copies of vital programs and data using cartridges specifically reserved for this purpose.

If READ errors begin to occur frequently when using a tape cartridge, it is advisable that steps be taken to prevent the loss of information stored on the tape. The first step is to clean the tape as discussed previously in this section. If this does not alleviate the problem, the next step is to transfer the information to a new medium and retire the worn tape. Continued use could cause loss of information or damage to the tape drive itself.

STALL errors (signifying tape transport error caused by motor overload) can occur when either the tape drive or the cartridge itself fails. To determine the source of the problem, a different cartridge can be inserted. If STALL errors stop occuring, assume the cartridge itself is bad and replace it. If STALL errors continue to occur, the drive may require servicing. In this case, contact your HP dealer for assistance.

Tape cartridges that have reached the end of their useful life exhibit some specific danger signals:

1. The oxide starts breaking loose from the mylar backing of the magnetic tape.
2. The cartridge drive belt becomes loose, evidenced by the tape winding unevenly on the tape reels. This condition can be seen through the top of the cartridge. (Slight unevenness is common; you should be concerned when the tape is uneven by a quarter of the width of the tape.)
3. The drive pulley of the tape cartridge contains dark spots due to slippage. In severe cases, the cartridge may stall and the capstan will wear a flat spot on the drive pulley.



4. The cartridge rattles rather than making a constant hum when any tape movements occur.
5. You begin to get recurring READ errors or STALL errors.

If any of the above five danger signals occur, you should replace the cartridge at once. If you continue to use a cartridge under these circumstances, there is a chance that you could lose all the information on your cartridge and that you could damage the tape transport itself.

CAUTION

Ignoring READ, STALL, EOF, or SEARCH errors in ON ERROR routines is not recommended. These errors can signify tape transport problems. Overriding any of them could easily damage the transport.

Tape Cartridge Rethreading

If the tape runs off of the cartridge reel, it either signifies a tape transport problem or the light path in the cartridge is being obstructed. Do not block the light window of the cartridge, because the tape will not operate properly. Tape rethreading is difficult and is not recommended unless the data recorded on the runoff tape must be recovered. Instead, if tape runoff occurs, it is recommended to replace the entire tape cartridge. The rethreading procedures contained in this paragraph are for rethreading tape onto the tape cartridge's left tape hub. If a tape runoff condition occurs from the right tape hub, use the left-hand instructions and change all counterclockwise directions to clockwise directions. This procedure requires the use of a small Pozidrive screwdriver. Rethread tape onto the left tape hub as follows:

CAUTION

Whenever the tape cartridge top cover is removed, the spring-loaded door and spring can easily slide off the door pivot post. To prevent loss of parts, ensure that door is always completely seated on its pivot post as long as the tape cartridge top cover and backplate are separted.

a. Remove tape cartridge top cover by removing four screws from backplate with Pozidrive (small Phillips-head) screwdriver.

b. As shown in figure A, rethread loose end of tape around right tape guide, past belt drive pulley, outside front guide pin, and around left tape guide so that approximately 1-3/4 inches of tape is clear of guide.

c. Hold tape cartridge as shown in figure B, so that right hand can be used to rotate belt drive pulley and left hand can be used to maintain tape tension at tape guide.

d. Moisten inside surface of free end of tape and, while maintaining tape tension at left tape guide, rotate belt drive pulley counterclockwise to wrap free end of tape around left tape hub until tape reaches point where drive belt touches tape hub.

e. While maintaining tape tension, use any small round-tipped tool to trap free end of tape between drive belt and left tape hub as shown in figure C.

f. Rotate belt drive pulley counterclockwise until tape is wrapped several times around left tape hub past first set of tape holes (approximately 2 feet). Check the tape pulleys to be sure they are not riding up.

g. Replace tape cartridge top cover on backplate and secure in place with four screws.

h. Condition tape in accordance with the instructions contained under Tape Care (page 282).

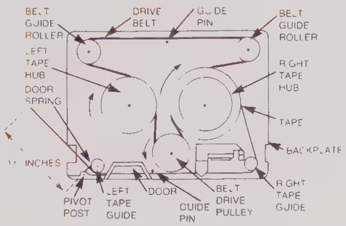

BELT GUIDE ROLLER
DRIVE BELT
GUIDE PIN
BELT GUIDE ROLLER
LEFT TAPE HUB
RIGHT TAPE HUB
DOOR SPRING
TAPE
BACKPLATE
1¾ INCHES
PIVOT POST
LEFT TAPE GUIDE
DOOR
GUIDE PIN
BELT DRIVE PULLEY
RIGHT TAPE GUIDE

Figure A

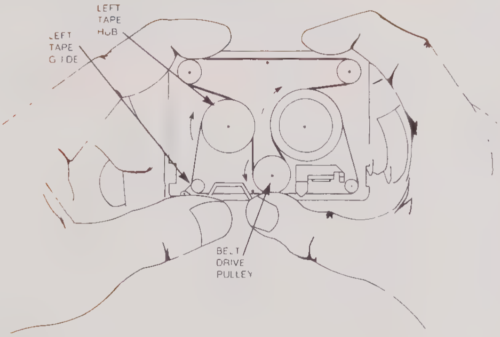

LEFT TAPE HUB
LEFT TAPE GUIDE
BELT DRIVE PULLEY

Figure B

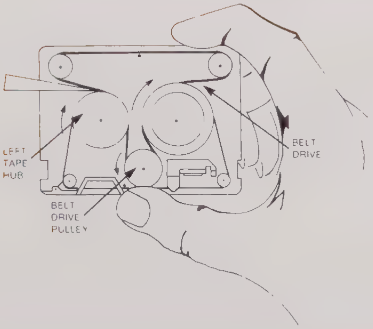
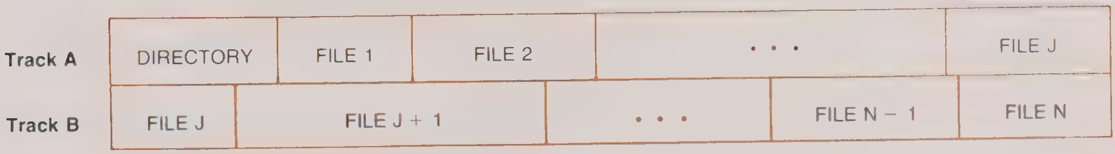

LEFT TAPE HUB
BELT DRIVE
BELT DRIVE PULLEY

Figure C

Optimizing Tape Use

A tape cartridge has two tape tracks with a variable number of records available in consecutively numbered files on each track, depending on the nature of your program and data storage requirements. The first file on track A and the first file on track B are both at the same end of the tape. This can cause a situation in which one file spans two tracks, making access to this file both time-consuming and wearing to the tape.



When this happens, it is a good idea to first label the file spanning both tracks as a "dummy" and then store the program or data again, using the following procedure:

```
        LOAD "file"
RENAME "file" TO "DUMMY"
       STORE "file"
```

The file named DUMMY will then span both tracks and will not be accessed. However, the STORE command causes the same program or data to be stored under the desired file name on the second track, immediately after the end of file DUMMY. Now, when accessing the program or data material in this file, the time loss and additional wear to the tape cartridge caused by running back and forth between tape tracks is avoided.

Operational Considerations

General Cleaning

Disconnect the HP-85 from its ac power source before cleaning.

The HP-85 can be cleaned with a soft cloth dampened either in clean water or in water containing a mild detergent. Do not use an excessively wet cloth, nor allow water inside the computer. Do not use any abrasive cleaners, especially on the tape cartridge window or the CRT screen.

The tape head should be cleaned after a maximum of 8 hours of use; refer to Tape Care, page 281

Selecting a Workspace

HP-85 computers are designed to operate on a flat, hard surface such as a desk or table top. Any workspace you choose for your HP-85 should allow the following minimum clearance dimensions for adequate air circulation around and within the instrument:

15 cm (6 in) both sides
15 cm (6 in) rear panel
15 cm (6 in) overhead

CAUTION

Always keep the top of the computer free of books, papers and other materials to avoid obstructing the air circulation vents built into the cover.

Potential for Radio/Television Interference

The HP-85 generates and uses radio frequency energy and may cause interference to radio and television reception. Your computer complies with the specifications in Subpart J of Part 15 of the FCC Rules for a Class B computing device. These specifications provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If the HP-85 does cause interference to radio or television reception, which can be determined by turning the computer off and on, you can try to eliminate the interference problem by doing one or more of the following:

- Reorient the receiving antenna.
- Change the position of the computer with respect to the receiver.
- Move the computer away from the receiver.
- Plug the computer into a different outlet so that the computer and the receiver are on different branch circuits.

If necessary, consult an authorized HP dealer or an experienced radio/television technician for additional suggestions. You may find the following booklet, prepared by the Federal Communications Commission, helpful: *How to Identify and Resolve Radio-TV Interference Problems.* This booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock No. 004-000-00345-4.

Temperature Ranges

Temperature ranges for the HP-85 computer are:

Operating	5° to 40°C	40° to 105°F
Storage	−40° to 65°C	−40° to 150°F

Service

If at any time you suspect that the computer is malfunctioning, the following information will help you determine whether or not servicing is needed. If you are not familiar with the first part of this appendix, review it before proceeding with this section.

Display

If the CRT display blanks out or becomes erratic, or if the computer fails to respond to keyboard commands, turn the computer off and check the following:

1. Ensure that the voltage selector switch is set to the correct nominal line voltage for your area (115 Vac or 230 Vac).
2. Ensure that the correct fuse is installed for the power supply in your area (750mA for 115 Vac; T400mA for 230 Vac); and that the fuse is intact.
3. Unplug the power cord and inspect the power contacts on both power cord and the computer. Clean them if necessary.
4. Make sure that the power cord is securely plugged into both the computer and an earth-grounded ac outlet.
5. Adjust the brightness control on the computer's rear panel for optimum display clarity.

If, after step 5, the display fails to respond properly, service is required. (Refer to warranty information on the following pages.)

Tape Drive

If a STALL error appears on the display, or if the tape transport fails to operate, check the following:

1. Remove and examine the tape cartridge for defects. If any are found, discard the cartridge.
2. Clean the tape head as described under Tape Operations earlier in this appendix.
3. Test the tape transport using a fresh tape cartridge.

If, after step 3, the tape transport fails to operate properly, servicing is required. (Refer to the following warranty information.)

Printer

If the thermal printer fails to operate properly, follow the procedures outlined under Printer Operation in this appendix. If the printer continues to malfunction, servicing is required. (Refer to the following warranty information.)

Internal Timer

The HP-85 internal timer was checked at the factory to meet an initial accuracy of within 1 second per hour. Because of the effects of temperatures variations, aging, shocks, and vibrations on its quartz-crystal time standard, the HP-85 timer accuracy may vary slightly.

Warranty Information

The complete warranty statement is included in the information packet shipped with your HP-85. Please retain this statement for your records.

If you have any questions concerning this warranty, **please contact the authorized HP-85 dealer or the HP sales and service office from which you purchased your HP-85.** Should you be unable to contact them, please contact:

In the U.S.:

Hewlett-Packard
Corvallis Division • Customer Support
1000 N.E. Circle Blvd.
Corvallis, Oregon 97330
Tel. (503) 757-1010

Toll Free Number (6 a.m. to 6 p.m., Pacific Time):
Call 800/547-3400 (except in Alaska and Hawaii).

In Europe:

Hewlett-Packard S.A.
7, rue du Bois-du-Lan
P.O. Box
CH-1217 Meyrin 2
Geneva
Switzerland

Tel. (022) 82 70 00

Other countries:

Hewlett-Packard Intercontinental
3495 Deer Creek Rd.
Palo Alto, California 94304
U.S.A.

Tel. (415) 856-1501

For world-wide HP sales and service offices, please refer to the back of the manual.

How to Obtain Repair Service

Not all Hewlett-Packard facilities offer service for the HP-85. For information on obtaining service in your area, consult the service information included in the information packet shipped with your HP-85. Or contact your authorized HP dealer or the nearest Hewlett-Packard sales and service facility. (Addresses are listed in the back of the manual.)

If your HP-85 malfunctions and repair is required, you can help assure efficient servicing by following these guidelines:

1. Leave configuration of the HP-85 exactly as it was at the time of the malfunction, i.e., any plug-in modules and tape cartridges in use at that time should remain intact.
2. Write a description of the malfunction symptoms for Service personnel.
3. Save printouts or any other materials that illustrate the problem area.
4. Have on hand a sales slip or other proof of purchase to establish warranty coverage period.

Serial Number

Each HP-85 computer carries an individual serial number on the plate in the upper right-hand corner of the rear panel. It is recommended that owners keep a separate record of this number. Should your unit be lost or stolen, the serial number is often necessary for tracing and recovery, as well as any insurance claims. Hewlett-Packard does not maintain records of individual HP-85 owner's names and unit serial numbers.

General Shipping Instructions

Should you ever need to ship your HP-85, be sure it is packed in a protective package (use the original shipping case), to avoid in-transit damage. Such damage is not covered by the warranty. Hewlett-Packard suggests that the customer always insure shipments.

Further Information

Computer design and circuitry are proprietary to Hewlett-Packard and service manuals are not available to customers.

Should other problems or questions arise regarding repairs, please call your nearest Hewlett-Packard sales and service facility or your authorized HP-85 dealer.

> **Note:** Not all Hewlett-Packard repair facilities offer service for HP-85 computers. However, you can be sure that service may be obtained in the country where you bought your computer.

If you happen to be outside of the country where you bought your computer, contact the nearest authorized HP-85 dealer or the local Hewlett-Packard center. All customs and duties are your responsibility.

Appendix C

Reference Tables

Reset Conditions

The following table shows the status of specific functions when the indicated commands are executed. Parentheses in the POWER ON column indicate the values when the system is turned on. "R" designates a function restored to POWER ON values. "–" designates a function unchanged from its status prior to executing the command. "U" designates that variables are assigned undefined values, except those in COMmon.

	POWER ON	RESET	SCRATCH	RUN	CHAIN	INIT	CONT
Program variables	R(none)	—	R	U	U	U	—
Calculator variables	R(none)	R	R	R	R	R	R
Result	R(zero)	R	R	—	—	—	—
Trigonometric Mode	R(RAD)	R	—	—	—	—	—
Typing Mode	R(BASIC)	—	—	—	—	—	—
PRINT ALL mode	R(off)	R	—	—	—	—	—
Output device	R(PRINTER IS 2 CRT IS 1)	R	—	—	—	—	—
Special function key definitions	R(none)	R	R	R	R	R	—
DATA pointers	R(none)	R	R	R	R	R	—
Default values	R(DEFAULT ON)	R	—	—	—	—	—
System timer							
TIME	R(zero)	—	—	—	—	—	—
DATE	R(zero)	—	—	—	—	—	—
Random Number Seed	R	R	—	—	—	—	—
KEY LABEL	R(none)	R	R	R	R	R	—
ON TIMER	R(off)	R	R	R	R	R	—
ON ERROR	R(off)	R	R	R	R	R	—
TRACE	R(off)	R	R	—	—	—	—
TRACE VAR	R(off)	R	R	—	U	—	—
TRACE ALL	R(off)	R	R	—	—	—	—
Binary programs	R(none)	—	R	—	—	—	—
SCALE	R(0,100,0,100)	R	—	—	—	—	—
ROMS	R(initialize)	R	—	—	—	—	—
PEN	R(positive)	R	—	—	—	—	—
PENUP	R(up)	R	—	—	—	—	—
Last plotted point	R(0,0)	R	—	—	—	—	—
LDIR	R(horizontal)	R	—	—	—	—	—
ASSIGN# buffers	R(none)	R	R	R	—	R	—
COMmon variables	R(none)	—	R	R	—	R	—

HP-85 Character and Key Codes

A numeric code is attached to each character and/or key. The first column of characters in the table below may be accessed by holding down the CTRL key while typing the character denoted by the superscript "c." Five characters in the last column are not apparent from the keyboard. They may be accessed by holding down the SHIFT key while typing the character or key denoted by the superscript "s."

In addition, each of the characters in the table below has a complementary underscored character with a decimal-value of 128 larger than its given decimal value. The `CHR$` function and certain keys on `INPUT` enable you to access the underscored characters. For instance `CHR$(74+128)` is J̲.

	EQUIVALENT FORMS				EQUIVALENT FORMS				EQUIVALENT FORMS				EQUIVALENT FORMS		
Char.	Binary	Octal	Dec	Char.	Binary	Octal	Dec	Char.	Binary	Octal	Dec	Char.	Binary	Octal	Dec
# @^{c}	00000000	000	0	SPACE	00100000	040	32	@	01000000	100	64	` KEY LABEL^s	01100000	140	96
… A^c	00000001	001	1	!	00100001	041	33	A	01000001	101	65	a	01100001	141	97
· B^c	00000010	002	2	"	00100010	042	34	B	01000010	102	66	b	01100010	142	98
Ñ C^c	00000011	003	3	#	00100011	043	35	C	01000011	103	67	c	01100011	143	99
ñ D^c	00000100	004	4	$	00100100	044	36	D	01000100	104	68	d	01100100	144	100
Ë E^c	00000101	005	5	%	00100101	045	37	E	01000101	105	69	e	01100101	145	101
Γ F^c	00000110	006	6	&	00100110	046	38	F	01000110	106	70	f	01100110	146	102
ä G^c	00000111	007	7	'	00100111	047	39	G	01000111	107	71	g	01100111	147	103
Δ H^c	00001000	010	8	(	00101000	050	40	H	01001000	110	72	h	01101000	150	104
ï I^c	00001001	011	9	)	00101001	051	41	I	01001001	111	73	i	01101001	151	105
¨ J^c	00001010	012	10	*	00101010	052	42	J	01001010	112	74	j	01101010	152	106
à K^c	00001011	013	11	+	00101011	053	43	K	01001011	113	75	k	01101011	153	107
ü L^c	00001100	014	12	,	00101100	054	44	L	01001100	114	76	l	01101100	154	108
CR M^c	00001101	015	13	-	00101101	055	45	M	01001101	115	77	m	01101101	155	109
⊤ N^c	00001110	016	14	.	00101110	056	46	N	01001110	116	78	n	01101110	156	110
ǂ O^c	00001111	017	15	/	00101111	057	47	O	01001111	117	79	o	01101111	157	111
Ä P^c	00010000	020	16	0	00110000	060	48	P	01010000	120	80	p	01110000	160	112
ü Q^c	00010001	021	17	1	00110001	061	49	Q	01010001	121	81	q	01110001	161	113
å R^c	00010010	022	18	2	00110010	062	50	R	01010010	122	82	r	01110010	162	114
Å S^c	00010011	023	19	3	00110011	063	51	S	01010011	123	83	s	01110011	163	115
ö T^c	00010100	024	20	4	00110100	064	52	T	01010100	124	84	t	01110100	164	116
Ö U^c	00010101	025	21	5	00110101	065	53	U	01010101	125	85	u	01110101	165	117
ï V^c	00010110	026	22	6	00110110	066	54	V	01010110	126	86	v	01110110	166	118
ë W^c	00010111	027	23	7	00110111	067	55	W	01010111	127	87	w	01110111	167	119
ù X^c	00011000	030	24	8	00111000	070	56	X	01011000	130	88	x	01111000	170	120
Ü Y^c	00011001	031	25	9	00111001	071	57	Y	01011001	131	89	y	01111001	171	121
Ö Z^c	00011010	032	26	:	00111010	072	58	Z	01011010	132	90	z	01111010	172	122
Æ [^{c}	00011011	033	27	;	00111011	073	59	[	01011011	133	91	π /^{s}	01111011	173	123
æ \\$^c$	00011100	034	28	<	00111100	074	60	\	01011100	134	92	¦	01111100	174	124
ø]^c	00011101	035	29	=	00111101	075	61	]	01011101	135	93	→ −^{s}	01111101	175	125
£ ∧^{c}	00011110	036	30	>	00111110	076	62	^	01011110	136	94	Σ *^{s}	01111110	176	126
▒ _^{c}	00011111	037	31	?	00111111	077	63	_	01011111	137	95	⊦ +^{s}	01111111	177	127

Key Response During Program Execution

Decimal codes above 128 are assigned to program, editing, and system control keys. The table below describes the response of the system when the specified key is pressed during the execution of a running program and its response to an `INPUT` statement.

Key Codes

Key	Response in ALPHA Mode	Response in Graphics Mode	Decimal Value
k1			128
k2			129
k3			130
k4			131
k5			132
k6			133
k7			134
k8			135
REWIND			136
COPY	A/L	A/L	137
PAPER ADVANCE	A/L	A/L	138
RESET	A/L	A/L	139
INIT			140
RUN			141
PAUSE	A/L	A/L	142
CONT			143
STEP			144
TEST			145
CLEAR	A/L	A/L	146
GRAPH	A/L	A/L	147
LIST			148
PLIST			149
KEY LABEL	A/L	A/L	150
not used			151
not used			152
BACK SPACE	A	A	153
ENDLINE	A	A	154
SHIFT BACK SPACE	A		155
Cursor left	A		156
Cursor Right	A		157
Roll up	A/L	A/L	158
Roll down	A/L	A/L	159
-Line	A		160
Cursor up	A		161
Cursor down	A		162
INS/RPL	A		163
-CHR	A		164
Cursor home	A		165
RESULT	A		166
not used			167
DELETE			168
STORE			169
LOAD			170
not used			171
AUTO			172
SCRATCH			173

L indicates that the specified key is live (i.e., performs its expected function) during the execution of a running program. All other keys halt a running program and then perform the indicated function.

A indicates that the specified key is active on `INPUT`. In other words, when the input prompt (?) appears, the keys designated by A perform their respective functions. All other keys output their respective character codes.

Appendix D

Glossary and BASIC Syntax Guidelines

The HP-85 BASIC language consists of **statements, functions, operators,** and **commands**. Operators and functions are used with variables, numbers, and strings to create numeric and string **expressions**. Expressions and functions can be included in statements and executed from the keyboard. Each statement can be preceded by a statement number and stored as a program statement. Most functions, statements, and commands can also be separately executed from the keyboard; exceptions are noted.

Operators

Arithmetic

+	Add
−	Subtract
*	Multiply
/	Divide
^	Exponentiate
`MOD`	Modulo: `A MOD B=A-B*INT(A/B)`
\ or `DIV`	Integer divide: `A DIV B=IP(A/B)`

Logical Evaluation

Logical expressions return the values 0 for false and 1 for true. Non-zero values are considered true; zero values are false.

Relational

=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<> or #	Not equal to

Non-numeric values can also be compared with relational operators. Strings are compared, character by character, from left to right until a difference is found. If one string ends before a difference is found, the shorter string is considered the lesser.

Logical

`AND`
`OR`
`EXOR`
`NOT`

String

`&` String concatenation

Math Hierarchy

Performed First ↓

`( )`

Functions

`^`

`NOT`

`*`, `/`, `MOD`, `\` or `DIV`

`+`, `-`

Relational operators (`=`, `>`, `<`, `>=`, `<=`, `<>`, or `#`)

`AND`

`OR`, `EXOR`

Performed Last

Expressions are evaluated from left to right for operators at the same level. Operations within parentheses are performed first. Nested parentheses are evaluated inward out.

Data Precision

Precision	Accuracy	Range	Maximum array size with standard memory and no program.
`REAL`	12 Digits	±9.99999999999E±499	1800
`SHORT`	5 Digits	±9.9999E±99	3600
`INTEGER`	5 Digits	−99999 through 99999	4800

Special Characters

`@` Enables multi-statement lines.

```
100 CLEAR @ KEY LABEL
```

`!` Remarks follows.

```
110 DISP  C ! Display cost.
```

`?` `INPUT` prompt. Input items are expected.

Variables

Simple Numeric Variables: `A1`, `B`, `C3`.

The name consists of a letter or a letter and one digit. Real precision is assumed unless `SHORT` or `INTEGER` type is declared.

Arrays: `A1(50,5)`, `B(20,20)`, `C3(10)`.

The name consists of a letter or a letter and one digit. An array name can be the same as a simple variable name used elsewhere in the program, but a one-dimensional array cannot have the same name as a two-dimensional array. Arrays contain numeric elements only. Subscripts dimension the row or row and column in `DIM`, `COM`, or type (`REAL`, `INTEGER`, `SHORT`) declaration statements. The lower bound of an array subscript is 0 unless `OPTION BASE 1` is specified before all array references. The default upper bound for row and column subscripts is 10.

Subscripts reference a particular array element in non-declaratory statements with three exceptions. Entire arrays (either one- or two-dimensional) may be referenced in TRACE VAR, PRINT#, or READ# statements by specifying the array name followed by a pair of parentheses and no subscripts (e.g., C3( )). A comma may be enclosed within the parentheses for documentation purposes to specify a two-dimensional array (e.g., A1(,)). This notation enables you to trace, write onto tape, or read from tape all elements of the specified array.

String Variables: A1$, B$, C3$.

The name consists of a letter or a letter and one digit followed by a dollar sign. The default length is 18 characters unless otherwise specified in a COM or DIM statement. The maximum length of a string is limited only by available memory. Dimension strings in a DIM or COM statement by specifying the variable name followed by the length enclosed within brackets: A1$[25], B$[415], C3$[5].

Substrings: A1$[2,25], B$[5], C3$[3,3].

Substrings are specified by one or two numbers (or expressions) enclosed within *brackets*. One number specifies a beginning character; the substring extends to the end of the string. Two numbers separated by a comma specify beginning and ending character positions, respectively.

Strings can be compared with the relational operators and can be concatenated by & operator. (Refer to Operators.)

Syntax Guidelines to Commands and BASIC Statements

These terms and conventions are used in the following list of statements and commands.

dot matrix	All items in dot matrix denote system commands or BASIC statements that must appear exactly as shown. Either small or capital letters may be used to spell keywords.
[ ]	All items enclosed within square brackets are optional unless the brackets are in dot matrix.
...	Three dots indicate that the previous item can be repeated.
statement number	An integer from 1 through 9999.
numeric expression	A logical combination of variables, constants, operators, and functions (including user-defined functions), grouped within parentheses as necessary.
string expression	A logical combination of text within quotes, string variables, substrings, string concatenations, and string functions.
file name or **program name**	A program or file name can be any string expression. Any letter, number, symbol, or character except quotes or the null string may be used. Only the first six letters of the string expression are used for the name.

```
LOAD "PROG1$"
LOAD T$
```

buffer number	The number assigned to a tape data file by an ASSIGN# statement, and referenced by the PRINT# and READ# statements. Its range is 1 through 10.

Commands

Non-Programmable

AUTO [beginning line number [, increment value]]	Page 80
CONT [statement number]	Page 98
DELETE first statement number [, last statement number]	Page 95
INIT	Page 99
LOAD program name	Page 179
REN [first statement number [, increment value]]	Page 96
RUN [statement number]	Page 99
SCRATCH	Page 78
STORE program name	Page 176
UNSECURE file name, security code, secure type	Page 194

Programmable

CAT	Page 175
COPY	Page 35
CTAPE	Page 282
ERASETAPE	Page 175
FLIP	Page 34
LIST [beginning statement number [, ending statement number]]	Page 97
PLIST [beginning statement number [, ending statement number]]	Page 97
PRINT ALL	Page 35
REWIND	Page 280
SECURE file name, security code, secure type	Page 193

BASIC Statements

ASSIGN# buffer number TO file name	Page 183
ASSIGN# buffer number TO *	Page 184
BEEP [tone, duration]	Page 89
CHAIN file name	Page 179
CLEAR	Page 19
COM common variable list	Page 123
CRT IS output code number	Page 169
CREATE file name, number of records [, number of bytes per record]	Page 180
DATA data list	Page 137
DEFAULT OFF	Page 70
DEFAULT ON	Page 70
DEF FN numeric variable name [(parameter)][= numeric expression]	Page 145
DEF FN string variable name [(parameter)][= string expression]	Page 145
DEG	Page 66
DIM dimension list	Page 121
DISP display list	Page 84

DISP USING image format string [; disp using list]	Page 167
DISP USING statement number [; disp using list]	Page 161
END	Page 77
FN END	Page 147
FOR loop counter = initial value TO final value [STEP increment value]	Page 111
GOSUB statement number	Page 151
GOTO statement number	Page 91
GRAD	Page 66
IF numeric expression THEN statement number or executable statement [ELSE statement number or executable statement]	Page 106
IMAGE image format string	Page 161
INPUT variable name$_1$ [, variable name$_2$ ...]	Page 87
INTEGER numeric variable [(subscripts)][, numeric variable [(subscripts)] ...]	Page 122
KEY LABEL	Page 154
[LET] numeric variable$_1$ [, numeric variable$_2$ ...] = numeric expression	Page 90
[LET] string variable$_1$ [, string variable$_2$ ...] = string expression	Page 90
[LET] FN variable name = expression	Page 147
LOAD BIN file name	Page 193
NEXT loop counter	Page 111
NORMAL	Pages 35, 80
OFF ERROR	Page 261
OFF KEY# key number	Page 156
OFF TIMER# timer number	Page 157
ON ERROR GOSUB statement number	Page 261
ON ERROR GOTO statement number	Page 261
ON numeric expression GOSUB statement number list	Page 153
ON numeric expression GOTO statement number list	Page 108
ON KEY# key number [, key label] GOSUB statement number	Page 154
ON KEY# key number [, key label] GOTO statement number	Page 154
ON TIMER# timer number , milliseconds GOSUB statement number	Page 156
ON TIMER# timer number , milliseconds GOTO statement number	Page 156
OPTION BASE 1 or 0	Page 121
PAUSE	Page 99
PRINT [print list]	Page 86
PRINT# buffer number ; print # list	Page 185
PRINT# buffer number , record number [; print # list]	Page 188
PRINT USING image format string [; print using list]	Page 167
PRINT USING statement number [; print using list]	Page 161
PRINTER IS output code number	Page 169
PURGE file name [, purge code number]	Page 192
RAD	Page 66
RANDOMIZE [numeric expression]	Page 64
READ variable name$_1$ [, variable name$_2$ ...]	Page 137

READ# buffer number ; variable list	Page 187
READ# buffer number , record number [; variable list]	Page 190
REAL numeric variable [(subscripts)][, numeric variable [(subscripts)] ...]	Page 122
REM [any combination of characters]	Page 83
RENAME old file name TO new file name	Page 192
RESTORE [statement number]	Page 139
RETURN	Page 151
SETTIME seconds since midnight , Julian day in form yyddd	Page 56
SHORT numeric variable [(subscripts)][, numeric variable [(subscripts)] ...]	Page 122
STOP	Page 77
STORE BIN file name	Page 193
TRACE	Page 255
TRACE ALL	Page 256
TRACE VAR variable$_1$ [, variable$_2$ ...]	Page 255
WAIT number of milliseconds	Page 100

Graphics Statements

ALPHA	Page 197
BPLOT character string, number of characters per line	Page 237
DRAW x-coordinate , y-coordinate	Page 211
GCLEAR [y]	Page 199
GRAPH	Page 197
IDRAW x-increment , y-increment	Page 217
IMOVE x-increment , y-increment	Page 217
LABEL character string	Page 221
LDIR numeric expression	Page 224
MOVE x-coordinate , y-coordinate	Page 211
PEN numeric expression	Page 207
PENUP	Page 207
PLOT x-coordinate , y-coordinate	Page 208
SCALE xmin , xmax , ymin , ymax	Page 199
XAXIS y-intercept [, tic length [, xmin , xmax]]	Page 202
YAXIS x-intercept [, tic length [, ymin , ymax]]	Page 202

BASIC Predefined Functions

ABS(X)	Absolute value of X.	Page 60
ACS(X)	Arcosine of X, in 1st or 2nd quadrant.	Page 66
ASN(X)	Arcsine of X, in 1st or 4th quadrant.	Page 66
ATN(X)	Arctangent of X, in 1st or 4th quadrant.	Page 66
ATN2(Y,X)	Arctangent of Y/X, in proper quadrant.	Page 67
CEIL(X)	Smallest integer >=X.	Page 61
CHR$(X)	Character whose decimal character code is X, 0<=X <=255.	Page 131
COS(X)	Cosine of X.	Page 66
COT(X)	Cotangent of X.	Page 66

CSC(X)	Cosecant of X.	Page 66
DATE	Julian date in format yyddd (assumes system timer has been set properly).	Page 57
DTR(X)	Degree to radian conversion.	Page 67
EPS	Smallest positive machine number (1E-499).	Page 64
ERRL	Line number of latest error.	Page 261
ERRN	Number of latest error.	Page 261
EXP(X)	e^x	Page 65
FLOOR(X)	Same as INT(X) (relates to CEIL).	Page 61
FP(X)	Fractional part of X.	Page 60
INF	Largest machine number (9.99999999999E499).	Page 64
INT(X)	Largest integer <=X.	Page 61
IP(X)	Integer part of X.	Page 60
LEN(S$)	Length of string S$.	Page 128
LGT(X)	Log to the base 10 of X, X>0.	Page 65
LOG(X)	Natural logaritm, X>0.	Page 65
MAX(X,Y)	If X>Y then X, else Y.	Page 62
MIN(X,Y)	If X<Y then X, else Y.	Page 62
NUM(S$)	Decimal character code of first character of S$.	Page 132
PI	3.14159265359	Page 63
POS(S1$, S2$)	Searches string S1$ for the first occurrence of string S2$. Returns starting index if found, otherwise returns 0.	Page 129
RMD(X,Y)	Remainder of X/Y: X − Y * IP(X/Y).	Page 62
RND	Next number, X, in a sequence of pseudo-random numbers, 0 < = X < 1.	Page 64
RTD(X)	Radian to degree conversion.	Page 67
SEC(X)	Secant of X.	Page 66
SGN(X)	The sign of X, −1 if X<0, 0 if X=0, and +1 if X>0.	Page 62
SIN(X)	Sine of X.	Page 66
SQR(X)	Positive square root of X.	Page 62
TAB(N)	Skips to specified column.	Page 168
TAN(X)	Tangent of X.	Page 66
TIME	Time in seconds since midnight (assumes system timer has been set properly).	Page 57
UPC$(S$)	Returns string with all lower-case alphabetic characters converted to upper-case.	Page 133
VAL(S$)	Returns the numeric equivalent of the string S$.	Page 130
VAL$(X)	String equivalent of X.	Page 131

Appendix E

Error Messages

Error Number	Error Condition	Default values (errors 1-8 only) with DEFAULT ON
	Math Errors (1 thru 13)	
1	Underflow: expression underflows machine	0
2	Overflow: • Expression overflows machine • Attempt to store value >99999 or <−99999 in INTEGER variable. • Attempt to store value >9.9999E99 or <−9.9999E99 in SHORT variable.	±9.99999999999E499 ±99999 ±9.9999E99
3	COT or CSC of n*180°; n=integer.	9.99999999999E499
4	TAN or SEC of n*90°; n=odd integer.	9.99999999999E499
5	Zero raised to negative power.	9.99999999999E499
6	Zero raised to zero power.	1
7	Null data: • Uninitialized string variable, or missing string function assignment. • Uninitialized numeric variable, or missing numeric function assignment.	"" 0
8	Division by zero.	±9.99999999999E499
9	Negative value raised to non-integer power.	Remaining errors are non-defaultable.
10	Square root of negative number.	
11	Argument (parameter) out of range: • ATN2(0,0). • ASN or ACSN (-1<n<+1). • ON expression GOTO/GOSUB; expression of range. >1 or greater	
12	Logarithm of zero.	
13	Logarithm of negative number.	
14	Not used.	
	System Errors (15 thru 25)	
15	System error; correct by reloading program, pressing RESET, or turning system off, then on again.	
16	Continue before run; program not allocated.	
17	FOR nesting too deep; more than 255 active FOR-NEXT loops.	
18	GOSUB nesting too deep; more than 255 nested subroutines.	

Error Number	Error Condition
19	Memory overflow: • Attempting to RUN a program that requires more than given memory. • Attempting to edit too large a program; delete a nonexisting line to deallocate program, then edit. • Attempting to load a program larger than available memory. • Attempting to open a file with no available buffer space. • Attempting any operation that requires more memory than available. • Attempting to load or run a large program after a ROM has been installed. ROMs use up a certain amount of memory. Refer to the appropriate ROM manual.
20	Not used.
21	ROM missing; attempting to RUN program that requires ROM. An attempt to edit program with missing ROM will usually SCRATCH memory.
22	Attempt to edit, list, store, or overwrite a SECUREd program.
23	Self-test error; system needs repair.
24	Too many (more than 14) ROMS.
25	Two binary programs; attempting to load a second binary program into memory (only one binary program allowed in memory at any time).
26 thru 29	Not used.
	Program Errors (30 thru 57)
30	OPTION BASE error: • Duplicate OPTION BASE declaration. • OPTION BASE after array declaration. • OPTION BASE parameter not 0 or 1.
31	CHAIN error; CHAIN to a program other than BASIC main program: e.g., CHAINing to a binary program.
32	COMmon variable mismatch.
33	DATA type mismatch: • READ variable and DATA type do not agree. • READ# found a string but required a number.
34	No DATA to read: • READ and DATA expired. • RESTORE executed with no DATA statement.
35	Dimensioned existing variable; attempt to dimension a variable that has been previously declared or used. Move DIM statement to beginning of program and try again.
36	Illegal dimension: • Illegal dimension in default array declaration. • Array dimensions don't agree; e.g., referencing A(2) when A(5,5) is dimensioned or referencing A(0) when OPTION BASE 1 declared.
37	Duplicate user-defined function.
38	Function definition within function definition; needs FN END.
39	Reference to a nonexistent user-defined function: • Finding FN END with no matching DEF FN. • Exiting a function that was not entered with a function call after branching to the middle of a multi-line function.

Error Number	Error Condition
40	Illegal function parameter; function parameter mismatch (e.g., declared as string, called as numeric).
41	FN=; user-defined function assignment. Function assignment does not occur between DEF FN and FN END.
42	Recursive user-defined function.
43	Numeric input wanted.
44	Too few inputs. Less items were given than requested by an INPUT statement.
45	Too many inputs. More items were given than requested by an INPUT statement.
46	NEXT missing; FOR with no matching NEXT.
47	FOR missing; NEXT with no matching FOR.
48	END statement necessary.
49	Null data; uninitialized data.
50	Binary program missing; attempting to RUN program that requires binary program. An attempt to edit will usually SCRATCH memory.
51	RETURN without GOSUB reference.
52	Illegal IMAGE format string; unrecognized character in IMAGE.
53	Illegal PRINT USING: ● Data overflows IMAGE declaration. ● Numeric data with string IMAGE. ● String data with numeric IMAGE. ● PRINT USING image format string is not correct.
54	Illegal TAB argument. With DEFAULT ON, an illegal TAB argument gives a warning message and defaults to TAB(1).
55	Array subscript out of range.
56	String variable overflow; string too big for variable.
57	Missing line; reference to a nonexistent statement number.
58 thru 59	Not used.
	Tape Errors (60 thru 75)
60	Tape cartridge is write-protected; RECORD slide tab is in left-most position.
61	Attempting to create/record more than 42 files on tape.
62	Cartridge out when attempting tape operations.
63	Duplicate file name for RENAME or CREATE.
64	Empty file; attempting to access file that was never recorded (e.g., tape was ejected *before* program was stored but *after* name was written in directory). Refer to PURGE.

Error Number	Error Condition
65	End of tape: • Tape run-off; check cartridge. • Tape is full. • Not enough space to CREATE data file.
66	File closed: • Attempting READ#/PRINT# to file that has not been opened with ASSIGN#. • Attempting to close a closed file (warning only). • Tape has been ejected and reinserted.
67	File name: • Name does not exist when attempt to LOAD, ASSIGN#, LOAD BIN, PURGE, RENAME, or SECURE. • Name not in quotes. • Attempt to PURGE an open file.
68	File type mismatch: • Attempting to treat program as data file, or vice versa. • Attempting to treat binary program as BASIC main program file, or vice versa. • Attempting to treat data as binary program, or vice versa.
69	Random overflow; attempting to READ#/PRINT# beyond existing number of bytes in logically-defined record with random file access.
70	READ error; system cannot read tape.
71	End-of-File; no data beyond EOF mark in data file.
72	Record: • Attempting to READ#/PRINT# to record that doesn't exist; e.g., READ# 1,120 when only 100 records in file. • Attempting to READ#/PRINT# at end of file. • Lost in record: close file to release buffer.
73	Searches and does not find: • Bad tape cartridge; may have been exposed to magnetic field. • Cannot find directory, tape may need to be initialized.
74	Stall; either bad tape cartridge or transport problem, refer to Tape Operations, appendix B.
75	Not an HP-85 file; cannot read.
76 thru 79	Not used.
	Syntax Errors (80 thru 92)
80	Right parentheses,), expected.
81	Bad BASIC statement or bad expression. If it is an expression, try it again with DISP <expression> to get a better error message.
82	String expression error; e.g., right quote missing or null string given for file name.
83	Comma missing or more parameters expected (separated by commas).
84	Excess characters; delete characters at end of good line, then press END LINE.
85	Expression too big for system to interpret.

Error Number	Error Condition
86	Illegal statement after THEN
87	Bad DIM statement.
88	Bad statement: • COM in calculator mode. • User-defined function in calculator mode. • INPUT in calculator mode.
89	Invalid parameter: • ON KEY# less than 1 or greater than 8. • Attempt to TRACE a calculator mode variable. • PRINTER IS or CRT IS with invalid parameter. • CREATE with invalid parameters. • ASSIGN#, PRINT#, or READ# with buffer number other than 1 through 10. • Random READ# to record 0. • SETTIME with illegal time parameter. • ON TIMER#, OFF TIMER# with number other than 1, 2, or 3. • SCALE with invalid parameters. • AUTO or REN with invalid parameters. • LIST with invalid parameters. • DELETE with invalid parameters. • VAL$ with non-numeric parameter. • Any statement, command, or function for which parameters are given but they are invalid.
90	Line number too large; greater than 9999.
91	Missing parameter; e.g., DELETE with missing or invalid parameters.
92	Syntax error. Cursor returns to character where error was found.

Sample Solutions to Problems

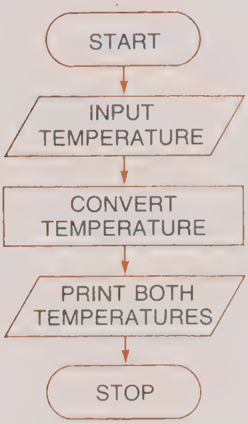
Section 5

Problem 5.1

(a)

```
10 REM *CELSIUS TO FAHRENHEIT
20 DISP "CELSIUS TEMP";
30 INPUT C
40 LET F=1.8*C+32
50 PRINT C;"C  EQUALS  ";F;"F"
60 END
```

Flowchart:


START
INPUT TEMPERATURE
CONVERT TEMPERATURE
PRINT BOTH TEMPERATURES
STOP

Display:

```
CELSIUS TEMP?
15
CELSIUS TEMP?
-10
```

Printer:

```
 15 C   EQUALS    59 F

-10 C   EQUALS    14 F
```

(b)

```
10 REM *FAHRENHEIT TO CELSIUS
20 DISP "FAHRENHEIT TEMP";
30 INPUT F
40 C=5/9*(F-32)
50 PRINT F;"F  EQUALS  ";C;"C"
60 END
```

Display:

```
FAHRENHEIT TEMP?
59
FAHRENHEIT TEMP?
14
```

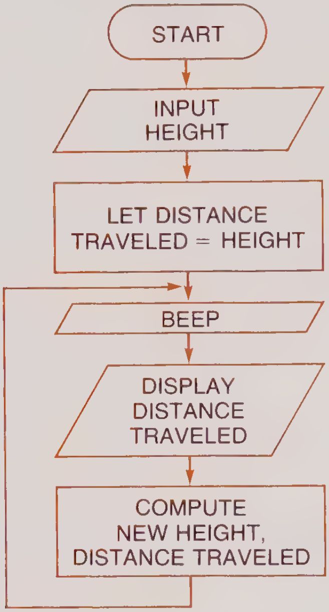
Printer:

```
59 F   EQUALS    15 C

14 F   EQUALS   -10 C
```

Problem 5.2

```
 10 REM *REBOUNDER
 20 DISP "HEIGHT RELEASED"
 30 INPUT H
 40 D=H
 50 BEEP
 60 DISP D
 70 H=.65*H
 80 D=D+2*H
 90 GOTO 50
100 END
```

Flowchart:


START
INPUT HEIGHT
LET DISTANCE TRAVELED = HEIGHT
BEEP
DISPLAY DISTANCE TRAVELED
COMPUTE NEW HEIGHT, DISTANCE TRAVELED

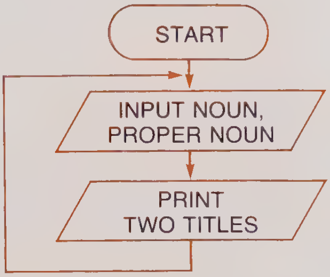
Display:

```
HEIGHT RELEASED
?
100
 100
 230
 314.5
 369.425
 405.12625
 428.3320625
 443.415840625
 453.220296406
 459.593192664
 463.735575232
 466.428123901
 468.178280536
 469.315882349
 470.055323527
```

Problem 5.3

```
10 REM *BOOK TITLES
20 DISP "NOUN, PROPER NOUN";
30 INPUT N$,P$
40 PRINT "THE ";N$;" OF ";P$
50 PRINT "TO ";P$;" WITH THE ";
   N$
60 GOTO 20
70 END
```

Flowchart:


START
INPUT NOUN, PROPER NOUN
PRINT TWO TITLES

Display:

```
NOUN, PROPER NOUN?
ANIMALS, AUSTRALIA
NOUN, PROPER NOUN?
ICEBERGS,ICELAND
NOUN, PROPER NOUN?
ATHLETES, THE OLYMPICS
NOUN, PROPER NOUN?
```

Printer:

```
THE ANIMALS OF AUSTRALIA
TO AUSTRALIA WITH THE ANIMALS
THE ICEBERGS OF ICELAND
TO ICELAND WITH THE ICEBERGS
THE ATHLETES OF THE OLYMPICS
TO THE OLYMPICS WITH THE
ATHLETES
```

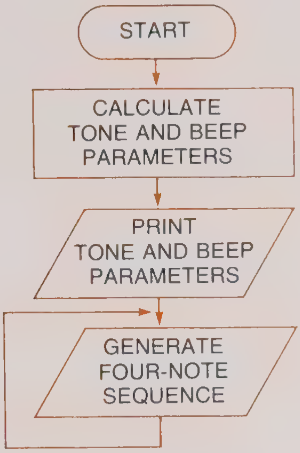
Problem 5.4

```
 10 REM *BASS RHYTHM
 20 F1=613062.5/(11*98)-134/11
 30 D1=.5*98
 40 F2=613062.5/(11*130.81)-134/
    11
 50 D2=.5*130.81
 60 F3=613062.5/(11*196)-134/11
 70 D3=.5*196
 80 DISP " F"," D",F1,D1,F2,D2,F
    3,D3
 90 BEEP F2,D2
100 BEEP F3,D3
110 BEEP F1,D1
120 BEEP F3,D3
130 GOTO 90
140 END
```

Display:

```
F                    D
556.521799629        49
413.878533056        65.405
272.169990723        98
```

Flowchart:


START
CALCULATE TONE AND BEEP PARAMETERS
PRINT TONE AND BEEP PARAMETERS
GENERATE FOUR-NOTE SEQUENCE

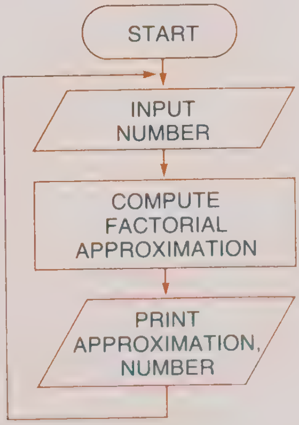
Problem 5.5

```
10 REM *FACTORIAL APPROX
20 DISP "X = ";
30 INPUT X
40 F=EXP(-X)*X^X*SQR(2*PI*X)
50 PRINT F,X;"!"
60 GOTO 20
70 END
```

Display:

```
X = ?
3
X = ?
6
X = ?
10
X = ?
50
X = ?
```

Flowchart:


START
INPUT NUMBER
COMPUTE FACTORIAL APPROXIMATION
PRINT APPROXIMATION, NUMBER

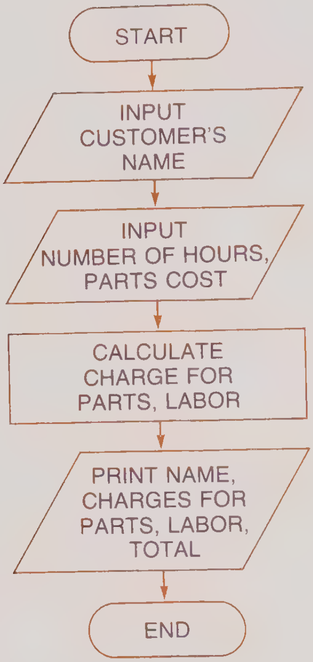
Printer:

```
5.83620959124        3 !
710.078184645        6 !
3598695.61874        10 !
3.03634459324E64     50 !
```

Problem 5.6

```
 10 REM *WATCH REPAIR
 20 DISP "CUSTOMER";
 30 INPUT N$
 40 DISP "HOURS WORKED, PARTS CO
    ST"
 50 INPUT H,P
 60 L=8.5*H
 70 P1=1.1*P
 80 PRINT
 90 PRINT "ITEMIZED REPAIR BILL:
      ";N$
100 PRINT "  PARTS          $";P1
110 PRINT "  LABOR          $";L
120 PRINT "TOTAL CHARGE  $";P1+L
130 END
```

Flowchart:


START
INPUT CUSTOMER'S NAME
INPUT NUMBER OF HOURS, PARTS COST
CALCULATE CHARGE FOR PARTS, LABOR
PRINT NAME, CHARGES FOR PARTS, LABOR, TOTAL
END

Display:

```
CUSTOMER?
WHIMPLE
HOURS WORKED, PARTS COST
?
2.5,12.00
```

Printer:

```
ITEMIZED REPAIR BILL:  WHIMPLE
  PARTS          $ 13.2
  LABOR          $ 21.25
TOTAL CHARGE     $ 34.45
```

Section 6

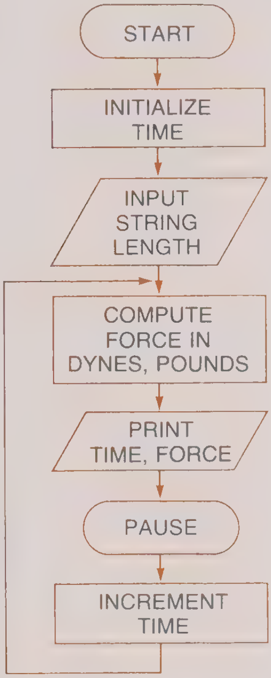
Problem 6.1

```
 10 REM *REBOUNDER
 20 DISP "HEIGHT RELEASED"
 30 INPUT H
 35 T=3000   <------------------------- Added line.
 40 D=H
 50 BEEP
 60 DISP D
 65 WAIT T   <------------------------- Added line.
 70 H=.65*H
 80 D=D+2*H
 85 T=.806*T <------------------------- Added line.
 90 GOTO 50
100 END
```

Problem 6.2

```
 10 REM *CENTRIFUGAL
 20 T=0
 30 DISP "STRING LENGTH";
 40 INPUT R
 50 F=350*(30*T)^2/R
 60 PRINT "SECONDS =";T
 70 PRINT "DYNES =";F
 80 PRINT "POUNDS =";.00000225*F
 90 PRINT
100 PAUSE
110 T=T+1
120 GOTO 50
130 END
```

Flowchart:


START
INITIALIZE TIME
INPUT STRING LENGTH
COMPUTE FORCE IN DYNES, POUNDS
PRINT TIME, FORCE
PAUSE
INCREMENT TIME

Display:

```
STRING LENGTH?
14
```

Printer:

```
SECONDS = 0
DYNES = 0
POUNDS = 0

SECONDS = 1
DYNES = 22500
POUNDS = .050625

SECONDS = 2
DYNES = 90000
POUNDS = .2025

SECONDS = 3
DYNES = 202500
POUNDS = .455625

SECONDS = 4
DYNES = 360000
POUNDS = .81

SECONDS = 5
DYNES = 562500
POUNDS = 1.265625

SECONDS = 6
DYNES = 810000
POUNDS = 1.8225

SECONDS = 7
DYNES = 1102500
POUNDS = 2.480625

SECONDS = 8
DYNES = 1440000
POUNDS = 3.24

SECONDS = 9
DYNES = 1822500
POUNDS = 4.100625

SECONDS = 10
DYNES = 2250000
POUNDS = 5.0625
```

Section 7

Problem 7.1

```
 10 REM *BASKETBALL
 20 A,W=0
 30 PRINT "  A    W"
 40 PRINT
 50 INPUT C$
 60 IF C$="A" THEN A=A+2
 70 IF C$="W" THEN W=W+2
 80 IF C$="a" THEN A=A+1
 90 IF C$="w" THEN W=W+1
100 PRINT A;W
110 GOTO 50
120 END
```

Problem 7.1 (Cont)

Display:

```
?
A
?
A
?
W
?
w
?
A
?
A
?
a
?
W
?
w
?
```

Printer:

```
A  W

2  0
4  0
4  2
4  3
6  3
8  3
9  3
9  5
9  6
```

Problem 7.2

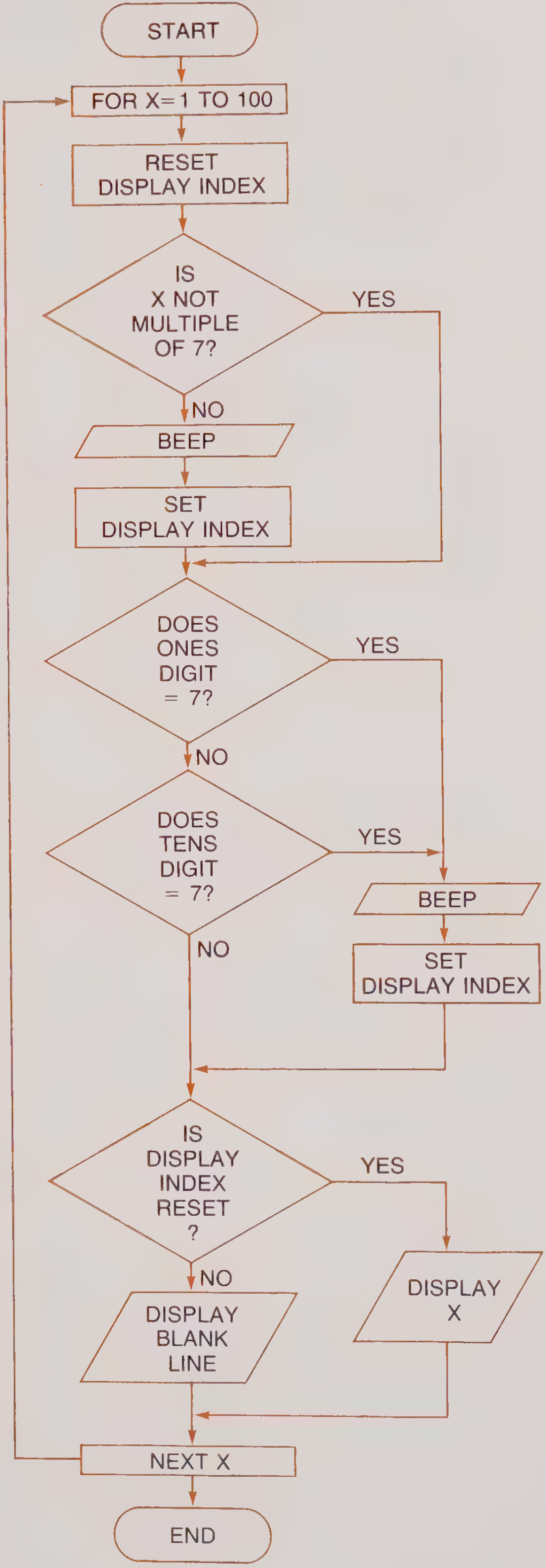
```
 10 REM *BEEP GAME
 20 FOR X=1 TO 100
 30 S=0
 40 IF FP(X/7)#0 THEN 70
 50 BEEP
 60 S=1
 70 IF 10*FP(X/10)=7 THEN 100
 80 IF IP(X/10)=7 THEN 100
 90 GOTO 120
100 BEEP
110 S=1
120 IF S=0 THEN DISP X ELSE DISP
130 NEXT X
140 END
```

Display:

```
1
2
3
4
5
6

8
9
10
11
12
13

15
16

18
19
.
.
.
```

Problem 7.2

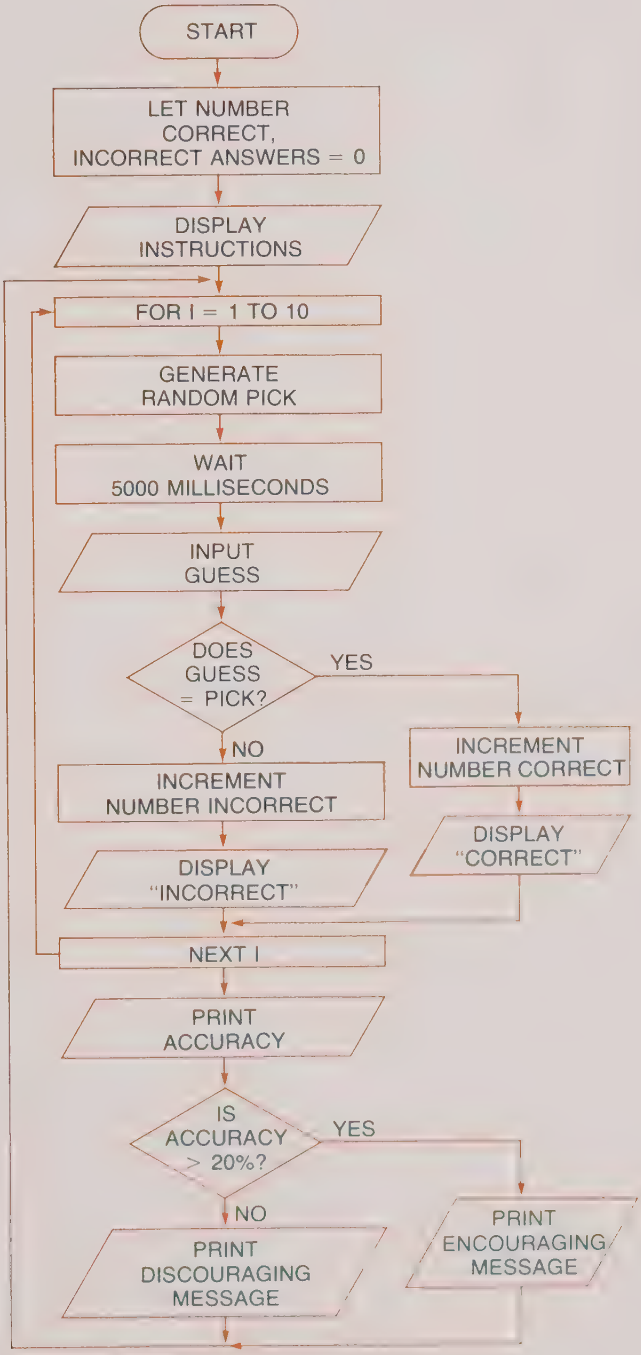
Flowchart:


START
FOR X=1 TO 100
RESET DISPLAY INDEX
IS X NOT MULTIPLE OF 7?
YES
NO
BEEP
SET DISPLAY INDEX
DOES ONES DIGIT = 7?
YES
NO
DOES TENS DIGIT = 7?
YES
NO
BEEP
SET DISPLAY INDEX
IS DISPLAY INDEX RESET ?
YES
NO
DISPLAY BLANK LINE
DISPLAY X
NEXT X
END

Problem 7.3

```
 10 REM *TELEPATHY
 20 R,W=0
 30 DISP "ENTER 1 TO 5 EACH TIME
    "
 40 FOR I=1 TO 10
 50 P=INT(1+5*RND)
 60 WAIT 5000
 70 INPUT N
 80 IF N=P THEN 170
 90 W=W+1
100 DISP "INCORRECT"
110 NEXT I
120 A=100*R/(R+W)
130 PRINT A;"% ACCURACY"
140 PRINT " FOR";R+W;"PICKS"
150 IF A>20 THEN PRINT "**TELEPA
    THY**" ELSE PRINT "JUST GUES
    SING"
160 GOTO 40
170 R=R+1
180 DISP "CORRECT"
190 GOTO 110
200 END
```

Flowchart:


START
LET NUMBER CORRECT, INCORRECT ANSWERS = 0
DISPLAY INSTRUCTIONS
FOR I = 1 TO 10
GENERATE RANDOM PICK
WAIT 5000 MILLISECONDS
INPUT GUESS
DOES GUESS = PICK?
YES
NO
INCREMENT NUMBER INCORRECT
DISPLAY "INCORRECT"
INCREMENT NUMBER CORRECT
DISPLAY "CORRECT"
NEXT I
PRINT ACCURACY
IS ACCURACY > 20%?
YES
NO
PRINT DISCOURAGING MESSAGE
PRINT ENCOURAGING MESSAGE

Display:

```
ENTER 1 TO 5 EACH TIME
?
2
INCORRECT
?
1
CORRECT
?
4
INCORRECT
?
5
INCORRECT
?
2
CORRECT
?
3
INCORRECT
?
5
INCORRECT
?
1
INCORRECT
?
2
INCORRECT
?
3
CORRECT
?
```

Printer:

```
 30 % ACCURACY
 FOR 10 PICKS
**TELEPATHY**
```

Problem 7.4

```
 10 REM *COMPASS COURSE
 20 DEG
 30 N,E=0
 40 DISP "BEARING, DISTANCE";
 50 INPUT B,D
 60 IF D=0 THEN 110
 70 N=N+D*COS(B)
 80 E=E+D*SIN(B)
 90 PRINT "BEAR";B;" DIST";D
100 GOTO 40
110 A=ATN2(E,N)
120 X=SQR(N*N+E*E)
130 IF A<0 THEN A=A MOD 360
140 PRINT "DIRECT ROUTE"
150 PRINT " BEARING ";A
160 PRINT " DISTANCE";X
170 PRINT
180 GOTO 30
190 END
```

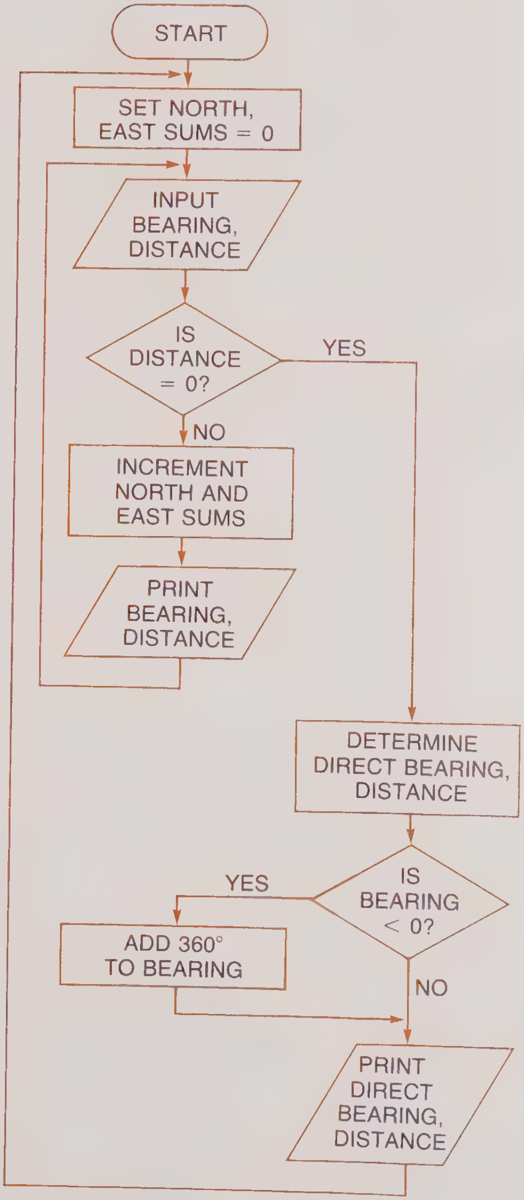
Display:

```
BEARING, DISTANCE?
85,350
BEARING, DISTANCE?
190,400
BEARING, DISTANCE?
265,250
BEARING, DISTANCE?
20,50
BEARING, DISTANCE?
0,0
BEARING, DISTANCE?
```

Printer:

```
BEAR 85  DIST 350
BEAR 190  DIST 400
BEAR 265  DIST 250
BEAR 20  DIST 50
DIRECT ROUTE
 BEARING  172.045343208
 DISTANCE 341.508929443
```

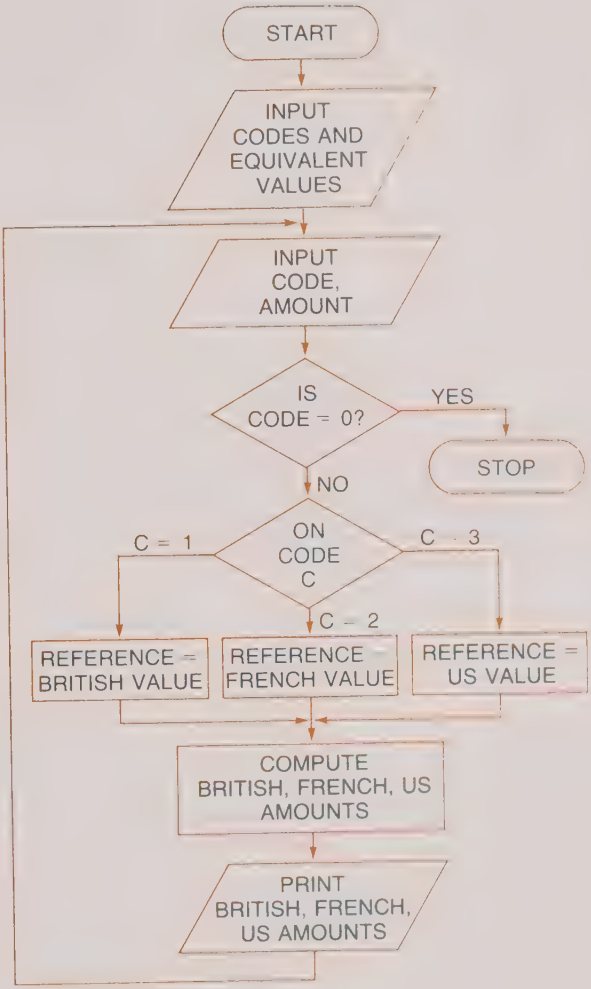
Flowchart:


START
SET NORTH, EAST SUMS = 0
INPUT BEARING, DISTANCE
IS DISTANCE = 0?
YES
NO
INCREMENT NORTH AND EAST SUMS
PRINT BEARING, DISTANCE
DETERMINE DIRECT BEARING, DISTANCE
IS BEARING < 0?
YES
ADD 360° TO BEARING
NO
PRINT DIRECT BEARING, DISTANCE

Problem 7.5

```
 10 REM *CURRENCY EXCHANGE
 20 DISP "ENTER CODE, AMT FOR 3
    SYSTS"
 30 DISP "1=BR, 2=FR,3=US"
 40 INPUT C1,A1,C2,A2,C3,A3
 50 DISP
 60 DISP "CODE, AMT TO CONVERT (
    0,0=STOP)"
 70 INPUT C,A
 80 IF C=0 THEN STOP
 90 REM *DETERMINE REFERENCE
100 ON C GOTO 110,130,150
110 R=A1
120 GOTO 160
130 R=A2
140 GOTO 160
150 R=A3
160 V1=A*A1/R
170 V2=A*A2/R
180 V3=A*A3/R
190 PRINT "EQUIVALENT AMOUNTS:"
200 PRINT "BR. POUND ";V1
210 PRINT "FR. FRANC ";V2
220 PRINT "US DOLLAR ";V3
230 PRINT
240 GOTO 50
250 END
```

Flowchart:


START
INPUT CODES AND EQUIVALENT VALUES
INPUT CODE, AMOUNT
IS CODE = 0?
YES
STOP
NO
ON CODE C
C = 1
C = 2
C = 3
REFERENCE = BRITISH VALUE
REFERENCE = FRENCH VALUE
REFERENCE = US VALUE
COMPUTE BRITISH, FRENCH, US AMOUNTS
PRINT BRITISH, FRENCH, US AMOUNTS

Display:

```
ENTER CODE, AMT FOR 3 SYSTS
1=BR, 2=FR,3=US
?
1,1,2,8.3981,3,1.8248

CODE, AMT TO CONVERT (0,0=STOP)
?
1,284

CODE, AMT TO CONVERT (0,0=STOP)
?
3,1205

CODE, AMT TO CONVERT (0,0=STOP)
?
0,0
```

Printer:

```
EQUIVALENT AMOUNTS:
BR. POUND  284
FR. FRANC  2385.0604
US DOLLAR  518.2432

EQUIVALENT AMOUNTS:
BR. POUND  660.346229325
FR. FRANC  5545.65459228
US DOLLAR  1205
```

Problem 7.6

```
 10 REM *DIMSBURG DELAY
 20 DISP "TOTAL DELAY IN MINUTES
    ";
 30 INPUT T
 40 IF T<0 THEN 80
 50 IF T>=3 THEN 160
 60 I=IP(T)+1
 70 ON I GOTO 100,120,140
 80 P=0
 90 GOTO 170
100 P=T^3/6
110 GOTO 170
120 P=.5-T*(1.5-T*(1.5-T/3))
130 GOTO 170
140 P=-3.5+T*(4.5-T*(1.5-T/6))
150 GOTO 170
160 P=1
170 PRINT "FOR DELAY LESS THAN";
    T
180 PRINT "  PROBABILITY IS";P
190 PRINT
200 GOTO 20
210 END
```

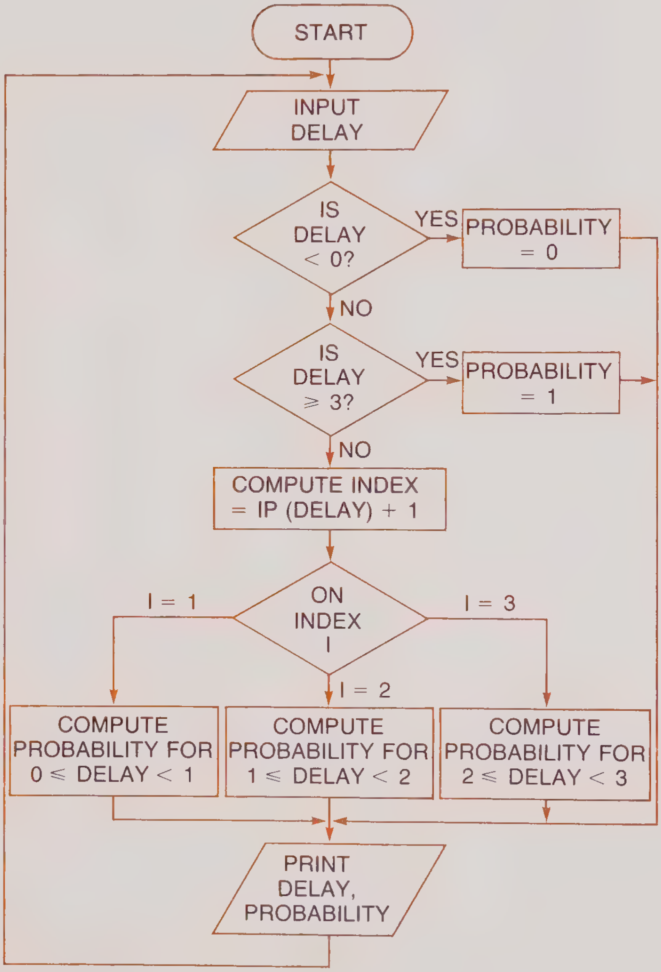
Display:

```
TOTAL DELAY IN MINUTES?
.5
TOTAL DELAY IN MINUTES?
1.5
TOTAL DELAY IN MINUTES?
2
TOTAL DELAY IN MINUTES?
```

Printer:

```
FOR DELAY LESS THAN .5
  PROBABILITY IS
 2.08333333333E-2

FOR DELAY LESS THAN 1.5
  PROBABILITY IS  5

FOR DELAY LESS THAN 2
  PROBABILITY IS .83333333332
```

Flowchart:


START
INPUT DELAY
IS DELAY < 0?
YES
PROBABILITY = 0
NO
IS DELAY ≥ 3?
YES
PROBABILITY = 1
NO
COMPUTE INDEX = IP (DELAY) + 1
ON INDEX I
I = 1
I = 2
I = 3
COMPUTE PROBABILITY FOR 0 ≤ DELAY < 1
COMPUTE PROBABILITY FOR 1 ≤ DELAY < 2
COMPUTE PROBABILITY FOR 2 ≤ DELAY < 3
PRINT DELAY, PROBABILITY

Section 8

Problem 8.1

```
 10 REM *INVENTAWORD
 20 DIM B$[20],F$[20],W$[21]
 30 DISP "BASE STRING";
 40 INPUT B$
 50 IF LEN(B$)=0 THEN STOP
 60 D$=B$[1,1]
 70 DISP "FIRST-LETTER STRING";
 80 INPUT F$
 90 IF LEN(F$)=0 THEN 30
100 FOR I=1 TO LEN(F$)
110 IF F$[I,I]=D$ THEN 140
120 W$=F$[I,I]&B$
130 PRINT W$
140 NEXT I
150 GOTO 70
160 END
```

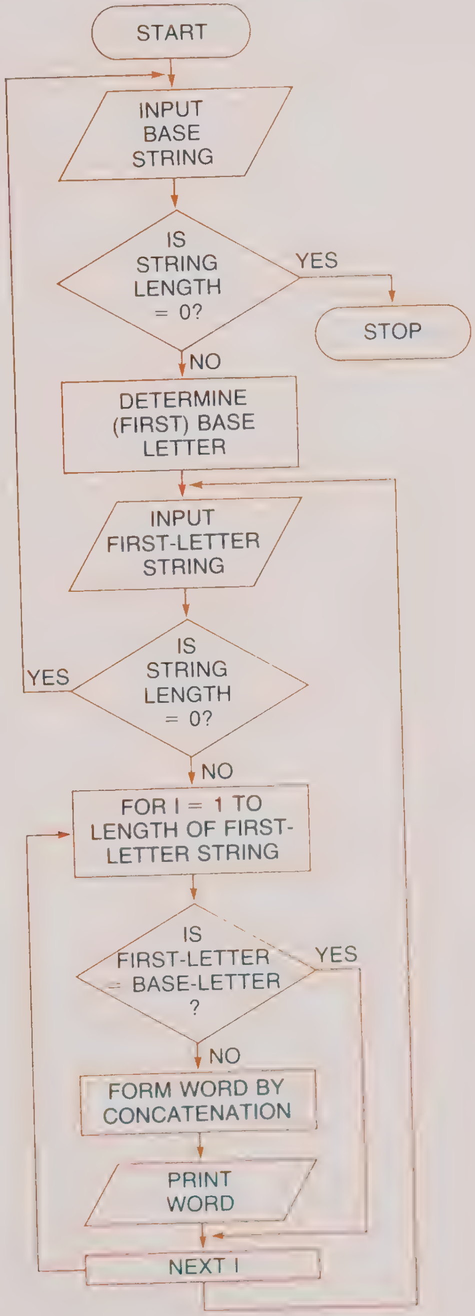
Display:

```
BASE STRING?
LUB
FIRST-LETTER STRING?
GBLF
FIRST-LETTER STRING?
OZ
FIRST-LETTER STRING?
```

Printer:

```
GLUB
BLUB
FLUB
OLUB
ZLUB
```

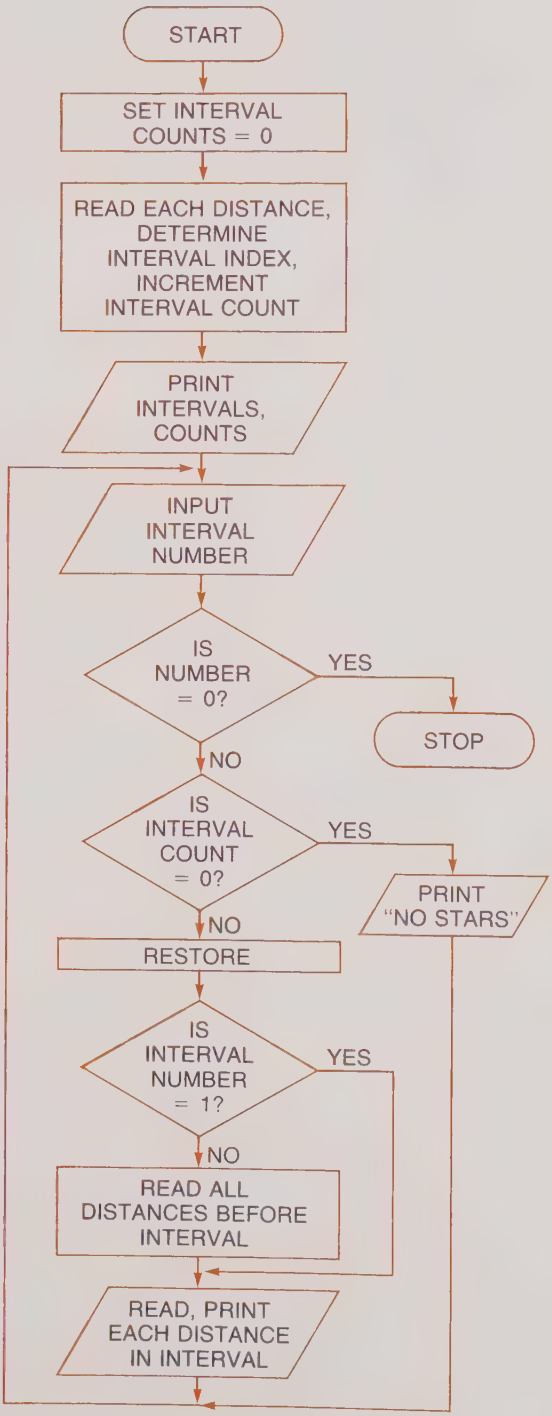
Flowchart:


START
INPUT BASE STRING
IS STRING LENGTH = 0?
YES
STOP
NO
DETERMINE (FIRST) BASE LETTER
INPUT FIRST-LETTER STRING
IS STRING LENGTH = 0?
YES
NO
FOR I = 1 TO LENGTH OF FIRST-LETTER STRING
IS FIRST-LETTER = BASE-LETTER ?
YES
NO
FORM WORD BY CONCATENATION
PRINT WORD
NEXT I

Problem 8.2

```
 10 REM *STAR DISTANCES
 20 OPTION BASE 1
 30 INTEGER N(15)
 40 SHORT D
 50 DATA 4.3,5.9,7.6,8.1,8.6,8.9
    ,9.4,10.3,10.7,10.8,10.8,11.
    2,11.2,11.4,11.5,11.6
 60 DATA 11.7,11.9,12.2,12.5,12.
    7,12.8,13.1,13.1,13.9,14.2,1
    4.5
 70 FOR I=1 TO 15
 80 N(I)=0
 90 NEXT I
100 FOR I=1 TO 27
110 READ D
120 J=IP(D)
130 N(J)=N(J)+1
140 NEXT I
150 PRINT "INTERVAL   STARS"
160 FOR I=1 TO 15
170 PRINT I-1;"-";I;"     ";N(I)
180 NEXT I
190 PRINT
200 DISP "INTERVAL";
210 INPUT K
220 IF K=0 THEN STOP
230 PRINT "INTERVAL";K-1;"-";K
240 IF N(K)=0 THEN 370
250 RESTORE
260 IF K=1 THEN GOTO 320
270 FOR I=1 TO K-1
280 FOR J=1 TO N(I)
290 READ D
300 NEXT J
310 NEXT I
320 FOR I=1 TO N(K)
330 READ D
340 PRINT D
350 NEXT I
360 GOTO 190
370 PRINT "  NO STARS"
380 GOTO 190
390 END
```

Flowchart:


START
SET INTERVAL COUNTS = 0
READ EACH DISTANCE, DETERMINE INTERVAL INDEX, INCREMENT INTERVAL COUNT
PRINT INTERVALS, COUNTS
INPUT INTERVAL NUMBER
IS NUMBER = 0?
YES
STOP
NO
IS INTERVAL COUNT = 0?
YES
PRINT "NO STARS"
NO
RESTORE
IS INTERVAL NUMBER = 1?
YES
NO
READ ALL DISTANCES BEFORE INTERVAL
READ, PRINT EACH DISTANCE IN INTERVAL

Display:

```
INTERVAL?
10
INTERVAL?
0
```

Printer:

```
INTERVAL   STARS
 0 - 1      0
 1 - 2      0
 2 - 3      0
 3 - 4      1
 4 - 5      1
 5 - 6      0
 6 - 7      1
 7 - 8      3
 8 - 9      1
 9 - 10      4
 10 - 11       7
 11 - 12       4
 12 - 13       3
 13 - 14       2
 14 - 15       0


INTERVAL 9 - 10
 10.3
 10.7
 10.8
 10.8
```

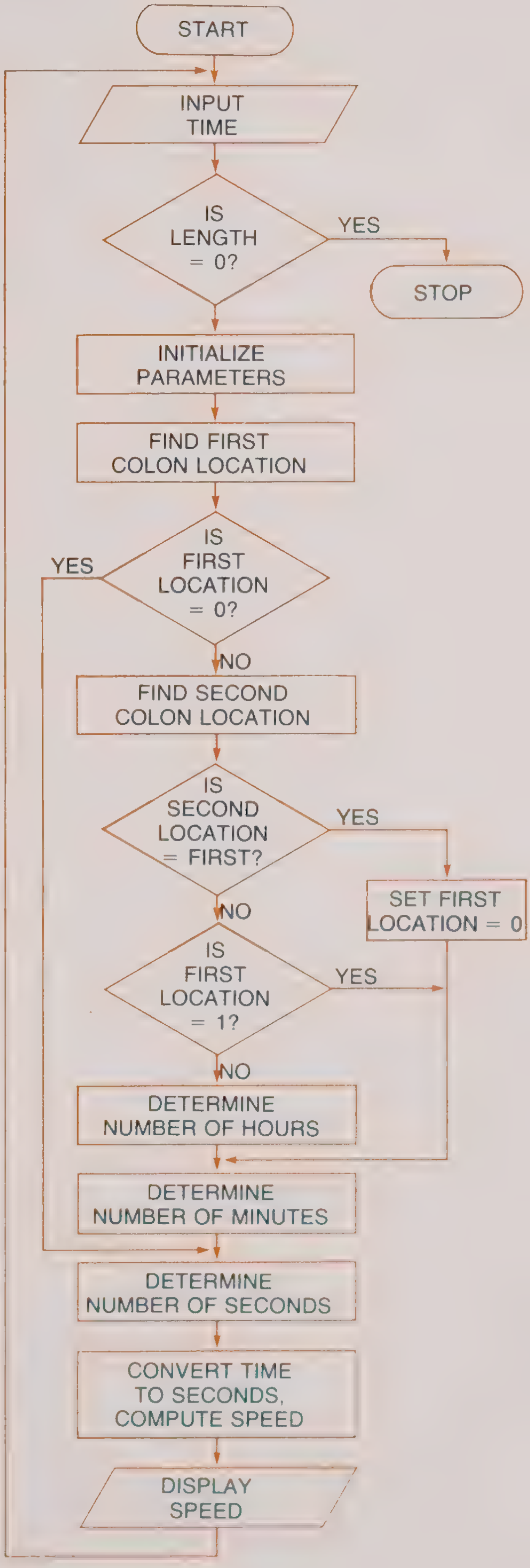
Problem 8.3

```
 10 REM *30-KM SPEEDS
 20 DISP "30-KM TIME",
 30 INPUT T$
 40 IF LEN(T$)=0 THEN STOP
 50 C1,C2=0
 60 H,M,S=0
 70 C1=POS(T$,":")
 80 IF C1=0 THEN 140
 90 C2=POS(T$[C1+1],":")+C1
100 IF C2=C1 THEN 190
110 IF C1=1 THEN 130
120 H=VAL(T$[1,C1-1])
130 M=VAL(T$[C1+1,C2-1])
140 S=VAL(T$[C2+1])
150 V=30000/(S+60*(M+60*H))
160 DISP "SPEED (m/s) =";V
170 DISP
180 GOTO 20
190 C1=0
200 GOTO 130
210 END
```

Display:

```
30-KM TIME?
1:31:30.4
SPEED (m/s) = 5.46408276264

30-KM TIME?
2:12:58.0
SPEED (m/s) = 3.76034093758

30-KM TIME?
1:30:29.38
SPEED (m/s) = 5.52549278186

30-KM TIME?
26:44
SPEED (m/s) = 18.7032418953

30-KM TIME?
```

Flowchart:


START
INPUT TIME
IS LENGTH = 0?
YES
STOP
INITIALIZE PARAMETERS
FIND FIRST COLON LOCATION
IS FIRST LOCATION = 0?
YES
NO
FIND SECOND COLON LOCATION
IS SECOND LOCATION = FIRST?
YES
SET FIRST LOCATION = 0
NO
IS FIRST LOCATION = 1?
YES
NO
DETERMINE NUMBER OF HOURS
DETERMINE NUMBER OF MINUTES
DETERMINE NUMBER OF SECONDS
CONVERT TIME TO SECONDS, COMPUTE SPEED
DISPLAY SPEED

Problem 8.4

```
 10 REM *COUNTING
 20 DIM W$[60]
 30 W$=" ZERO  ONE   TWO   THREE
    FOUR  FIVE  SIX   SEVEN EIG
   HT NINE "
 40 FOR I=0 TO 9
 50 FOR J=0 TO 9
 60 K=1+6*I
 70 IF I=0 THEN N$="" ELSE N$=W$
    [K,K+5]
 80 K=1+6*J
 90 N$=N$&W$[K,K+5]
100 DISP N$
110 NEXT J
120 NEXT I
130 END
```

Display:

```
ZERO
ONE
TWO
THREE
FOUR
FIVE
SIX
SEVEN
EIGHT
NINE
ONE    ZERO
ONE    ONE
ONE    TWO
ONE    THREE
ONE    FOUR
ONE    FIVE
```

Problem 8.5

```
 10 REM *SPRINKLER
 20 OPTION BASE 1
 30 DIM D(5,4)
 40 FOR I=1 TO 5
 50 READ D(I,1),D(I,2),D(I,3),D(
    I,4)
 60 DATA 124,133,138,142,126,136
    ,141,146,129,139,144,149
 70 DATA 132,142,147,152,134,145
    ,150,155
 80 NEXT I
 90 N$="ABCD"
100 PRINT " FT   PSI  NOZZLE"
110 DISP "WIDTH, PRESSURE";
120 INPUT W,P
130 PRINT
140 PRINT W;P;
150 J= MIN (5,IP(P/5-11))
160 IF J<1 THEN PRINT " TOO LOW"
     @ GOTO 110
170 FOR I=1 TO 4
180 IF W<=D(J,I) THEN 220
190 NEXT I
200 PRINT " TOO WIDE"
210 GOTO 110
220 PRINT " NOZZLE ";N$[I,I]
230 GOTO 110
240 END
```

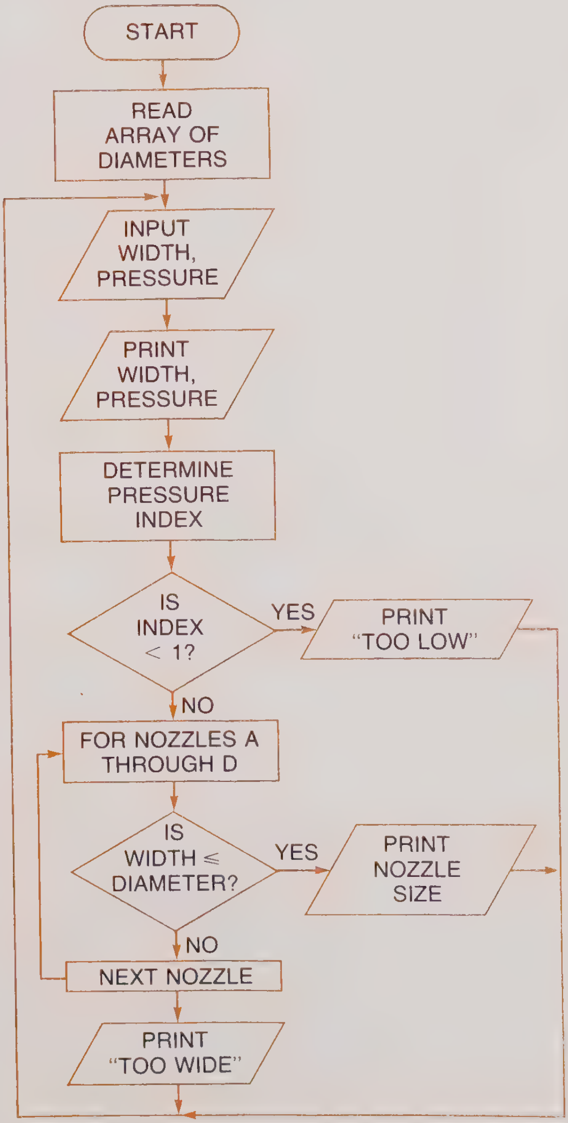
Display:

```
WIDTH, PRESSURE?
150,75
WIDTH, PRESSURE?
140,75
WIDTH, PRESSURE?
140,60
WIDTH, PRESSURE?
```

Printer:

```
FT    PSI   NOZZLE

150   75   NOZZLE D

140   75   NOZZLE B

140   60   NOZZLE D
```

Flowchart:


START
READ ARRAY OF DIAMETERS
INPUT WIDTH, PRESSURE
PRINT WIDTH, PRESSURE
DETERMINE PRESSURE INDEX
IS INDEX < 1?
YES
PRINT "TOO LOW"
NO
FOR NOZZLES A THROUGH D
IS WIDTH ≤ DIAMETER?
YES
PRINT NOZZLE SIZE
NO
NEXT NOZZLE
PRINT "TOO WIDE"

Section 9

Problem 9.1

(a)

```
10 REM *ROUND AT RADIX*
20 DEF FNR(D) = INT(D+.5)
30 FOR I=-5 TO 5 STEP .3
40 DISP I,FNR(I)
50 NEXT I
60 END
```

Display:

```
-5                   -5
-4.7                 -5
-4.4                 -4
-4.1                 -4
-3.8                 -4
-3.5                 -3
-3.2                 -3
-2.9                 -3
-2.6                 -3
-2.3                 -2
-2                   -2
-1.7                 -2
-1.4                 -1
-1.1                 -1
-.8                  -1
-.5                   0
-.2                   0
 .1                   0
 .4                   0
 .7                   1
 1                    1
 1.3                  1
 1.6                  2
 1.9                  2
 2.2                  2
 2.5                  3
 2.8                  3
 3.1                  3
 3.4                  3
 3.7                  4
 4                    4
 4.3                  4
 4.6                  5
 4.9                  5
```

(b)

```
10 REM *ROUND TO 3 DECIMAL PLAC
   ES
20 DEF FNR3(D) = INT(D*1000+.5)
   /1000
30 FOR I=1 TO 10 STEP .5
40 DISP I,FNR3(SQR(I))
50 NEXT I
60 END
```

Display:

```
1                    1
1.5                  1.225
2                    1.414
2.5                  1.581
3                    1.732
3.5                  1.871
4                    2
4.5                  2.121
5                    2.236
5.5                  2.345
6                    2.449
6.5                  2.55
7                    2.646
7.5                  2.739
8                    2.828
8.5                  2.915
9                    3
9.5                  3.082
10                   3.162
```

Problem 9.2

(a)

```
10 REM *AREA
20 DEF FNC(R) = PI*R*R
30 FOR I=350 TO 360
40 DISP I,FNC(I)
50 NEXT I
60 END
```

Display:

```
350
384845.100065
351
387047.356515
352
389255.896149
353
391470.718973
354
393691.824977
355
395919.214167
356
398152.886546
357
400392.842107
358
402639.080856
359
404891.602789
360
407150.407904
```

(b)

```
10 REM *ROUND AREA
20 DEF FNC(R) = PI*R*R
30 FOR I=350 TO 360
40 DISP I,FNR(FNC(I))
50 NEXT I
60 DEF FNR(N) = INT(N*100+.5)/1
   00
70 END
```

Display:

```
350                          384845.1
351                          387047.36
352                          389255.9
353                          391470.72
354                          393691.82
355                          395919.21
356                          398152.89
357                          400392.84
358                          402639.08
359                          404891.6
360                          407150.41
```

Problem 9.3

```
10 REM *FIND HYPOTENUSE
20 INPUT X
30 DEF FNA(X) = SQR(X*X+Y*Y)
40 FOR I=1 TO 5
50 INPUT Y
60 DISP FNA(X)
70 NEXT I
80 END
```

Problem 9.3 (Cont)

Display:

```
?
5
?
4
 6.40312423743
?
3
 5.83095169485
?
6
 7.81024967591
?
7
 8.60232526704
?
9
 10.295630141
```

Problem 9.4

(a)

```
 10 REM *OCTAL TO DECIMAL FN
 20 DEF FND(O)
 30 D=100000000000
 40 S=0
 50 X=IP(O/D)
 60 S=S*8+X
 70 O=O-X*D
 80 D=D/10
 90 IF D>=1 THEN 50
100 FND=S
110 FN END
```

(b)

```
 10 REM *OCTAL TO DECIMAL FN
 20 DEF FND(O)
 30 D=100000000000
 40 S=0
 50 X=IP(O/D)
 60 IF X>=8 OR X<0 THEN DISP "IN
    PUT POSITIVE OCTAL" @ FND=0
    @ GOTO 120
 70 S=S*8+X
 80 O=O-X*D
 90 D=D/10
100 IF D>=1 THEN 50
110 FND=S
120 FN END
130 DISP "INPUT OCTAL NUMBER";
140 INPUT I8
150 DISP FND(I8)
160 GOTO 140
170 END
```

Display:

```
INPUT OCTAL NUMBER?
200
 128
?
201
 129
?
208
INPUT POSITIVE OCTAL
 0
?
-10
INPUT POSITIVE OCTAL
 0
?
255
 173
?
217
 143
?
```

Problem 9.5

```
 10 REM *FACTORIAL FN
 20 INPUT X
 30 DISP X,FNF(X)
 40 GOTO 10
 50 DEF FNF(A)
 60 F=1
 70 FOR P=A TO 1 STEP -1
 80 F=F*P
 90 NEXT P
100 FNF=F
110 FN END
120 END
```

Display:

```
?
6
 6                       720
?
18
 18
 6.40237370574E15
?
12
 12                      479001600
?
10
 10                      3628800
?
```

Problem 9.6

```
 10 REM *ROUND TO 2 DECIMAL PLAC
    ES AND ADD '$'
 20 DEF FNR2$(D)
 30 N=INT(D*100+.5)/100
 40 IF FP(N)=0 THEN FNR2$="$"&VA
    L$(N)&".00" ELSE FNR2$="$"&V
    AL$(N)
 50 FN END
 60 DISP "INPUT NUMERIC VALUE";
 70 INPUT N
 80 DISP FNR2$(N)
 90 GOTO 70
100 END
```

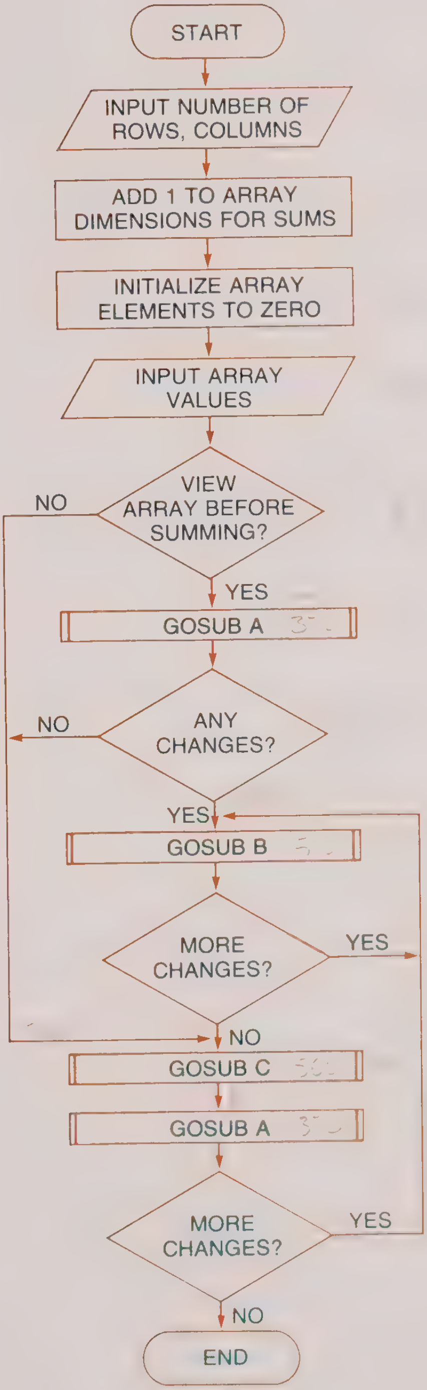
Display:

```
INPUT NUMERIC VALUE?
134.9876
$134.99
?
150
$150.00
?
```

Problem 9.7

```
 10 REM *ARRAY ROUTINES*
 20 OPTION BASE 1
 30 DIM A(10,5)
 40 DISP "MAX ARRAY SIZE IS 10 R
    OWS BY 5 COLUMNS"
 50 DISP "NUMBER ROWS,COLUMNS"
 60 INPUT R1,C1
 70 REM *INITIALIZE ARRAY
 80 R2=R1+1 @ C2=C1+1
 90 FOR R=1 TO R2
100 FOR C=1 TO C2
110 A(R,C)=0
120 NEXT C
130 NEXT R
140 REM *ENTER DATA, ONE VALUE A
    T A TIME, BY ROW.
150 FOR R=1 TO R1
160 DISP "INPUT DATA IN ROW";R
170 FOR C=1 TO C1
180 INPUT A(R,C)
190 NEXT C
200 NEXT R
210 DISP "DO YOU WANT TO SEE THE
     DATA TABLE BEFORE SUMMING (
    Y OR N)";
220 INPUT A$
230 IF A$="N" THEN 310
240 GOSUB 370
250 DISP "ANY CHANGES (Y OR N)";
260 INPUT B$
270 IF B$="N" THEN 310
280 GOSUB 510
290 DISP "MORE CHANGES";
300 GOTO 260
310 GOSUB 560
320 GOSUB 370
330 DISP "CHANGES",
340 INPUT C$
350 IF C$="Y" THEN 280
360 STOP
370 REM *COPY ARRAY
380 DISP "COPY TABLE ON PRINTER
    OR DISPLAY (P OR D)";
390 INPUT Z$
400 IF Z$="P" THEN CRT IS 2
410 DISP
420 DISP
430 FOR R=1 TO R2
440 FOR C=1 TO C2
450 DISP A(R,C);
460 NEXT C
470 DISP
480 NEXT R
490 CRT IS 1
500 RETURN
510 REM *CHANGE ELEMENT
520 DISP "ENTER ROW, COLUMN, VAL
    UE"
530 INPUT R,C,V
540 A(R,C)=V
550 RETURN
560 REM *SUM EACH ROW AND PLACE
    SUM IN LAST COLUMN
570 FOR R=1 TO R1
580 FOR C=1 TO C1
590 A(R,C2)=A(R,C2)+A(R,C)
600 NEXT C
610 NEXT R
620 REM *SUM EACH COLUMN AND PLA
    CE SUM IN LAST ROW.
630 FOR C=1 TO C1
640 FOR R=1 TO R1
650 A(R2,C)=A(R2,C)+A(R,C)
660 NEXT R
670 NEXT C
680 REM *FIND SUM OF VALUES IN L
    AST ROW (OR COLUMN)
690 FOR C=1 TO C1
700 A(R2,C2)=A(R2,C2)+A(R2,C)
710 NEXT C
720 DISP "SUMMING COMPLETED"
730 RETURN
```

Flowchart:


START
INPUT NUMBER OF ROWS, COLUMNS
ADD 1 TO ARRAY DIMENSIONS FOR SUMS
INITIALIZE ARRAY ELEMENTS TO ZERO
INPUT ARRAY VALUES
VIEW ARRAY BEFORE SUMMING?
NO
YES
GOSUB A
ANY CHANGES?
NO
YES
GOSUB B
MORE CHANGES?
YES
NO
GOSUB C
GOSUB A
MORE CHANGES?
YES
NO
END

Problem 9.7 (Cont)

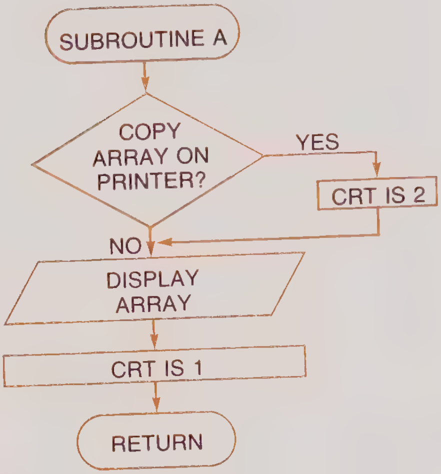
Display:

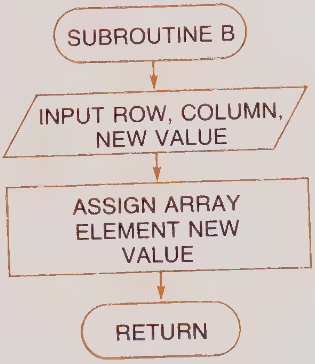
```
MAX ARRAY SIZE IS 10 ROWS BY 5 C
OLUMNS
NUMBER ROWS,COLUMNS
?
3,3
INPUT DATA IN ROW 1
?
12.59
?
13.69
?
14.78
INPUT DATA IN ROW 2
?
11.43
?
22.56
?
43.78
INPUT DATA IN ROW 3
?
13.52
?
12.78
?
14.98
DO YOU WANT TO SEE THE DATA TABL
E BEFORE SUMMING (Y OR N)?
Y
COPY TABLE ON PRINTER OR DISPLAY
 (P OR D)?
D


  12.59  13.69  14.78  0
  11.43  22.56  43.78  0
  13.52  12.78  14.98  0
  0  0  0  0
ANY CHANGES (Y OR N)?
Y
ENTER ROW, COLUMN, VALUE
?
1,3,14.67
MORE CHANGES?
N
SUMMING COMPLETED
COPY TABLE ON PRINTER OR DISPLAY
 (P OR D)?
P
CHANGES?
N
```

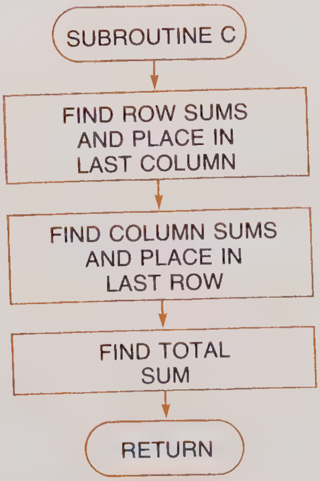
Printer:

```
  12.59  13.69  14.67  40.95
  11.43  22.56  43.78  77.77
  13.52  12.78  14.98  41.28
  37.54  49.03  73.43  160
```

Flowchart (subroutines):


SUBROUTINE A
COPY ARRAY ON PRINTER?
YES
CRT IS 2
NO
DISPLAY ARRAY
CRT IS 1
RETURN

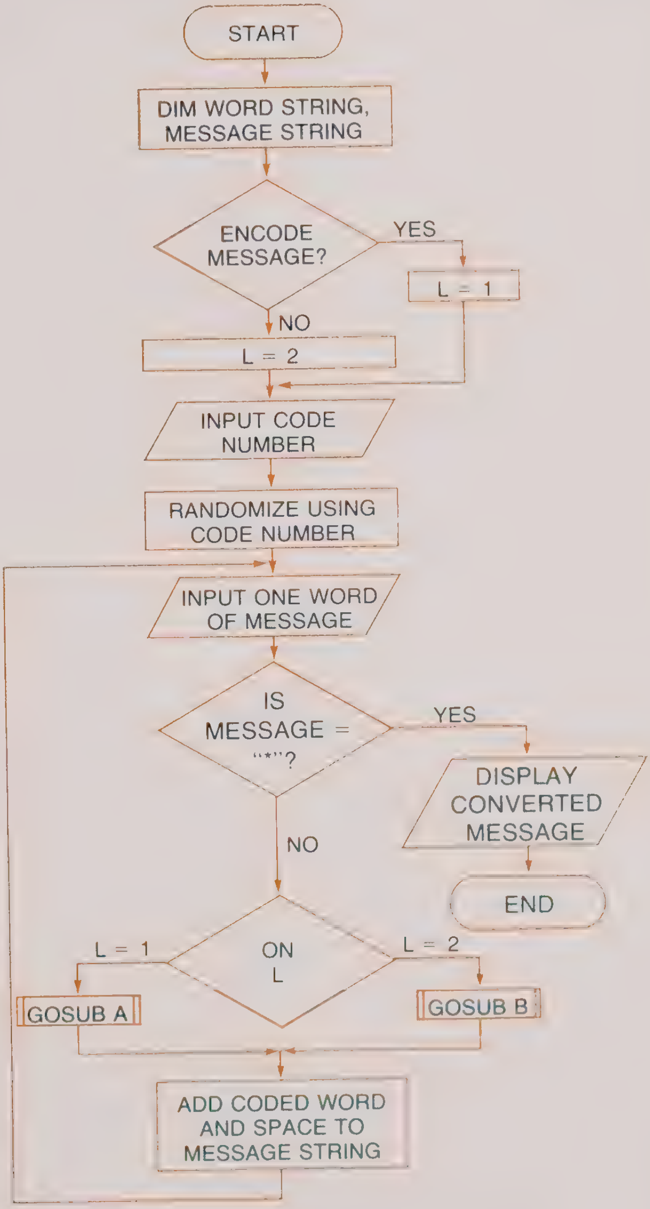

SUBROUTINE B
INPUT ROW, COLUMN, NEW VALUE
ASSIGN ARRAY ELEMENT NEW VALUE
RETURN


SUBROUTINE C
FIND ROW SUMS AND PLACE IN LAST COLUMN
FIND COLUMN SUMS AND PLACE IN LAST ROW
FIND TOTAL SUM
RETURN

Problem 9.8

```
 10 DIM I$[32],F$[1],M$[2000]
 20 DISP "CODE OR DECODE: C OR D
    "
 30 INPUT F$
 40 IF F$="C" THEN L=1 ELSE L=2
 50 DISP "CODE NUMBER PLEASE"
 60 INPUT S
 70 RANDOMIZE S
 80 M$=""
 90 DISP "TYPE MESSAGE ONE WORD
    AT A TIME. TYPE '*' TO END M
    ESSAGE"
100 DISP "GIVE ME YOUR MESSAGE"
110 INPUT I$
120 IF I$="*" THEN 160
130 ON L GOSUB 1000,2000
140 M$=M$&C$&" "
150 GOTO 110
160 DISP M$
170 END
1000 REM *ENCODING ROUTINE
1010 C$=""
1020 FOR I=1 TO LEN(I$)
1030 C$=C$&CHR$(65+(NUM(I$[I,I])
     +INT(26*RND)) MOD 26)
1040 NEXT I
1050 RETURN
2000 REM *DECODING ROUTINE
2010 C$=""
2020 FOR I=1 TO LEN(I$)
2030 C$=C$&CHR$(65+(NUM(I$[I,I])
     -INT(26*RND)) MOD 26)
2040 NEXT I
2050 RETURN
```

Flowchart:


START
DIM WORD STRING, MESSAGE STRING
ENCODE MESSAGE?
YES
L = 1
NO
L = 2
INPUT CODE NUMBER
RANDOMIZE USING CODE NUMBER
INPUT ONE WORD OF MESSAGE
IS MESSAGE = "*"?
YES
DISPLAY CONVERTED MESSAGE
END
NO
ON L
L = 1
L = 2
GOSUB A
GOSUB B
ADD CODED WORD AND SPACE TO MESSAGE STRING

Problem 9.8 (Cont)

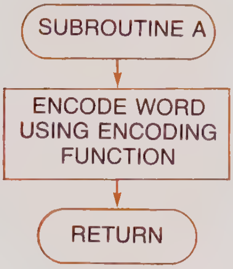
Display:

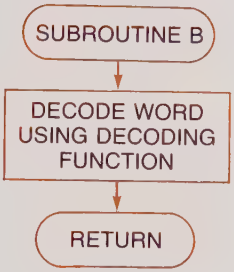
```
CODE OR DECODE: C OR D
?
C
CODE NUMBER PLEASE
?
.123
TYPE MESSAGE ONE WORD AT A TIME.
 TYPE '*' TO END MESSAGE
GIVE ME YOUR MESSAGE
?
GET
?
ME
?
TO
?
THE
?
BANK
?
ON
?
TIME
?
*
EWV SC PR YGB ZSMD NQ VWMS
```

```
CODE OR DECODE: C OR D
?
D
CODE NUMBER PLEASE
?
.123
TYPE MESSAGE ONE WORD AT A TIME.
 TYPE '*' TO END MESSAGE
GIVE ME YOUR MESSAGE
?
EWV
?
SC
?
PR
?
YGB
?
ZSMD
?
NQ
?
VWMS
?
*
GET ME TO THE BANK ON TIME
```

```
CODE OR DECODE: C OR D
?
D
CODE NUMBER PLEASE
?
.3579
TYPE MESSAGE ONE WORD AT A TIME.
 TYPE '*' TO END MESSAGE
GIVE ME YOUR MESSAGE
?
NNLSNUNUS
?
IGPXR
?
RQP
?
BVE
?
*
SYLVESTER WHERE ARE YOU
```

Flowchart (subroutines):


SUBROUTINE A
ENCODE WORD USING ENCODING FUNCTION
RETURN


SUBROUTINE B
DECODE WORD USING DECODING FUNCTION
RETURN

Problem 9.9

```
 10 REM *FN KEY ROUTINES*
 20 OPTION BASE 1
 30 DIM A(20,10)
 40 CLEAR
 50 ON KEY# 1,"INIT" GOSUB 120
 60 ON KEY# 2,"INPUT" GOSUB 230
 70 ON KEY# 3,"COPY-A" GOSUB 320
 80 ON KEY# 4,"CHANGE" GOSUB 460
 90 ON KEY# 5,"SUM" GOSUB 510
100 KEY LABEL
110 GOTO 110
120 DISP "NUMBER ROWS,COLUMNS"
130 INPUT R1,C1
140 REM *INITIALIZE ARRAY
150 R2=R1+1 @ C2=C1+1
160 FOR R=1 TO R2
170 FOR C=1 TO C2
180 A(R,C)=0
190 NEXT C
200 NEXT R
210 DISP "ARRAY INITIALIZED. NOW
     INPUT      YOUR DATA."
220 RETURN
230 REM *ENTER DATA, ONE VALUE A
    T A TIME, BY ROW.
240 FOR R=1 TO R1
250 DISP "INPUT DATA IN ROW";R
260 FOR C=1 TO C1
270 INPUT A(R,C)
280 NEXT C
290 NEXT R
300 DISP "ARRAY IS FILLED. DATA
    INPUT COMPLETE."
310 RETURN
320 REM *COPY ARRAY
330 DISP "COPY TABLE ON PRINTER
    OR DISPLAY (P OR D)";
340 INPUT Z$
350 IF Z$="P" THEN CRT IS 2
360 DISP
370 DISP
380 FOR R=1 TO R2
390 FOR C=1 TO C2
400 DISP A(R,C);
410 NEXT C
420 DISP
430 NEXT R
440 CRT IS 1
450 RETURN
460 REM *CHANGE ELEMENT
470 DISP "ENTER ROW, COLUMN, VAL
    UE"
480 INPUT R,C,V
490 A(R,C)=V
500 RETURN
510 REM *SUM EACH ROW AND PLACE
    SUM IN LAST COLUMN.
520 FOR R=1 TO R1
530 FOR C=1 TO C1
540 A(R,C2)=A(R,C2)+A(R,C)
550 NEXT C
560 NEXT R
570 REM *SUM EACH COLUMN AND PLA
    CE SUM IN LAST ROW
580 FOR C=1 TO C1
590 FOR R=1 TO R1
600 A(R2,C)=A(R2,C)+A(R,C)
610 NEXT R
620 NEXT C
630 REM *FIND SUM OF VALUES IN L
    AST ROW (OR COLUMN)
640 FOR C=1 TO C1
650 A(R2,C2)=A(R2,C2)+A(R2,C)
660 NEXT C
670 DISP "SUMMING COMPLETED"
680 RETURN
```

Array initialization routine on k1.

Input data routine on k2.

Display or print array on k3.

Change array element on k4.

Find sum of rows, columns, and total on k5.

```
--------------------------------
SUM
INIT      INPUT     COPY-A   CHANGE
```

The key labels appear on the display and the program waits for you to press a special function key. In this program, you must initialize the array before you do anything else. So, first press k1 and answer the question that appears on the display for summing the rows and columns of some tables of your own.

Section 10

Problem 10.1

```
 10 REM *NATIONAL SUMMARIES
 20 PRINT USING 30
 30 IMAGE "  POPULATION",5X,"ARE
    A   POP DENS"/
 40 PRINT USING 50
 50 IMAGE 7X,"ANNUAL GNP",5X,"GN
    P/PERS"/
 60 PRINT USING 70
 70 IMAGE 32("_")
 80 DISP "NATION"
 90 INPUT N$
100 IF LEN(N$)=0 THEN STOP
110 DISP "POP, AREA, GNP";
120 INPUT P,A,G
130 PRINT USING  "K,//" ; N$
140 PRINT USING 150 ; P;A;P/A
150 IMAGE X,2(3DC3DC3D),2X,DCDDZ
     .D/
160 PRINT USING 170 ; G;G/P
170 IMAGE "    $",DC3DC3DC3DC3D,"
      $",2DC3D//
180 GOTO 80
190 END
```

Display:

```
NATION
?
CHINA
POP, AREA, GNP?
865193550,9560980,223000000000
NATION
?
UNITED STATES
POP, AREA, GNP?
216817000,9363123,1781400000000
NATION
?
CANADA
POP, AREA, GNP?
23469142,9976139,195785000000
NATION
?
SINGAPORE
POP, AREA, GNP?
2322576,581,5885600000
NATION
?
MONGOLIA
POP, AREA, GNP?
1531940,1565000,547000000
NATION
?
QATAR
POP, AREA, GNP?
97792,11000,4044000000
NATION
?
```

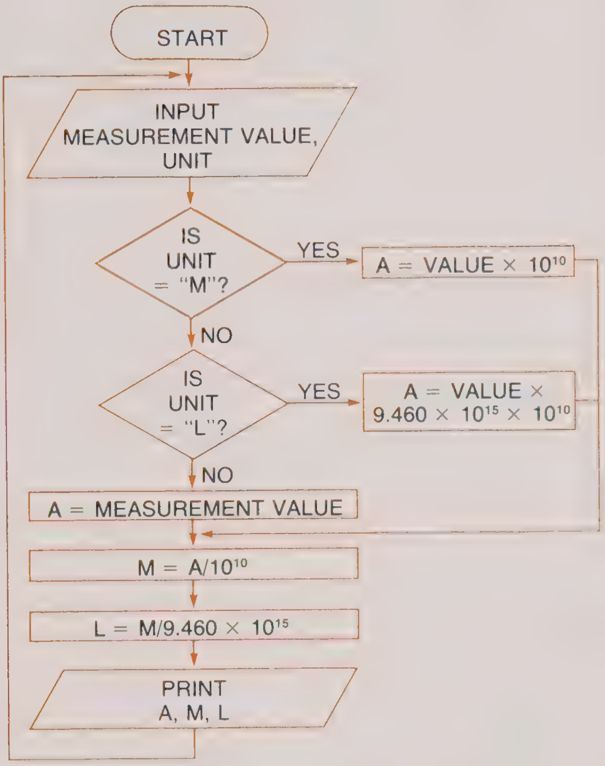
Printer:

```
  POPULATION     AREA   POP DENS

       ANNUAL GNP     GNP/PERS

________________________________
CHINA

 865,193,550  9,560,980     90.5

    $  223,000,000,000  $   258

UNITED STATES

 216,817,000  9 363,123     23.2

    $1,791,400,000,000  $ 8,216

CANADA

  23,469,142  9,976,139      2.4

    $  195,785,000,000  $ 8,342

SINGAPORE

   2,322,576        581  3,997.5

    $    5,885,600,000  $ 2,534

MONGOLIA

   1,531,940  1,565,000      1.0

    $      547,000,000  $   357

QATAR

      97,792     11,000      8.9

    $    4,044,000,000  $41,353
```

Problem 10.2

```
 10 REM *EXTREMES
 20 PRINT USING 30
 30 IMAGE "ANGSTROMS    METERS
    LIGHT-YEARS"
 40 PRINT USING 50
 50 IMAGE 32("_")
 60 DISP "UNITS: A,M,L"
 70 DISP "VALUE comma UNITS";
 80 INPUT X,U$
 90 IF U$="M" THEN 170
100 IF U$="L" THEN 190
110 A=X
120 M=A/10000000000
130 L=M/9.46E15
140 PRINT USING 150 ; A;M;L
150 IMAGE 3(D.3DE,X)
160 GOTO 70
170 A=X*10000000000
180 GOTO 120
190 A=X*9.46E15*10000000000
200 GOTO 120
210 END
```

Flowchart:


START
INPUT MEASUREMENT VALUE, UNIT
IS UNIT = "M"?
YES
A = VALUE × 10^10
NO
IS UNIT = "L"?
YES
A = VALUE × 9.460 × 10^15 × 10^10
NO
A = MEASUREMENT VALUE
M = A/10^10
L = M/9.460 × 10^15
PRINT A, M, L

Display:

```
UNITS: A,M,L
VALUE comma UNITS?
5560,A
VALUE comma UNITS?
1410,M
VALUE comma UNITS?
5 6E-3,A
VALUE comma UNITS?
1E-14,M
VALUE comma UNITS?
170000,L
VALUE comma UNITS?
```

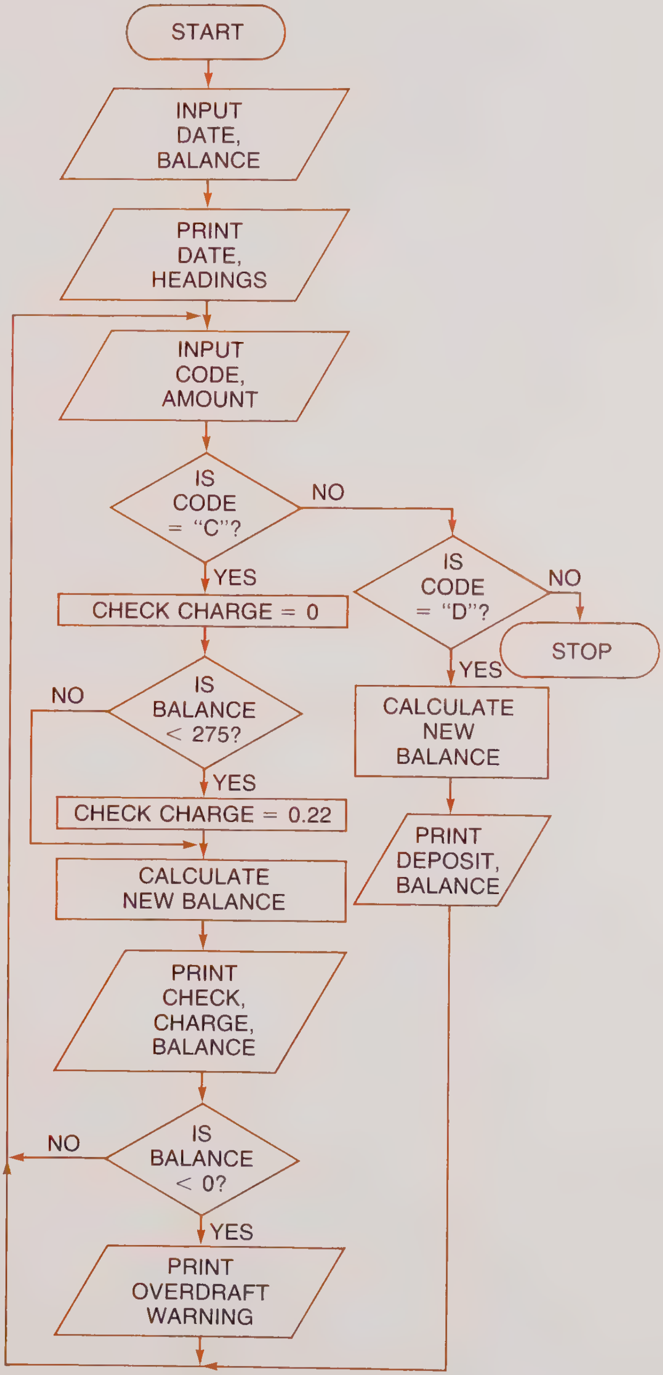
Printer:

```
ANGSTROMS    METERS   LIGHT-YEARS
--------------------------------
5 560E+003 5.560E-007 5 877E-023

1.410E+013 1.410E+003 1.490E-013

5 600E-003 5 600E-013 5 920E-029

1 000E-004 1 000E-014 1 057E-030

1 608E+031 1 608E+021 1 700E+005
```

Problem 10.3

```
 10 REM *CHECKING ACCT
 20 DISP "REFERENCE DATE";
 30 INPUT T$
 40 DISP "BEGINNING BALANCE";
 50 INPUT B
 60 PRINT "SUMMARY FOR ";T$
 70 PRINT USING 80
 80 IMAGE /" CHECKS   CHG DEPOSI
    TS   BALANCE"
 90 PRINT USING 100
100 IMAGE 32("_")
110 PRINT USING  "/24X,DCDDZ.DD"
     ; B
120 DISP "TRANSACTION (C,D), AMT
    ";
130 INPUT C$,A
140 IF C$="C" THEN 170
150 IF C$="D" THEN 250
160 STOP
170 F=0
180 IF B<275 THEN F=.22
190 B=B-A-F
200 PRINT USING 210 ; A;F;B
210 IMAGE DCDDZ.DD,2X,.DD,11X,DC
    DDZ.DD
220 IF B<0 THEN PRINT USING 230
    ; -B
230 IMAGE /"**WARNING   $",***Z.
    DD," OVERDRAFT**"/
240 GOTO 120
250 B=B+A
260 PRINT USING 270 ; A;B
270 IMAGE 12X,2(2X,DCDDZ.DD)
280 GOTO 120
290 END
```

Flowchart:


START
INPUT DATE, BALANCE
PRINT DATE, HEADINGS
INPUT CODE, AMOUNT
IS CODE = "C"?
NO
YES
CHECK CHARGE = 0
IS CODE = "D"?
NO
STOP
YES
CALCULATE NEW BALANCE
PRINT DEPOSIT, BALANCE
NO
IS BALANCE < 275?
YES
CHECK CHARGE = 0.22
CALCULATE NEW BALANCE
PRINT CHECK, CHARGE, BALANCE
NO
IS BALANCE < 0?
YES
PRINT OVERDRAFT WARNING

Display:

```
REFERENCE DATE?
JUNE 1979
BEGINNING BALANCE?
1027.41
TRANSACTION (C,D), AMT?
C,850.00
TRANSACTION (C,D), AMT?
C,54.79
TRANSACTION (C,D), AMT?
D,55.45
TRANSACTION (C,D), AMT?
C,185.49
TRANSACTION (C,D), AMT?
D,255.00
TRANSACTION (C,D), AMT?
```

Printer:

```
SUMMARY FOR JUNE 1979

 CHECKS   CHG DEPOSITS   BALANCE
--------------------------------

                        1,027.41
  850.00  .00             177.41
   54.79  .22             122.40
                 55.45    177.85
  185.49  .22           -   7.86

**WARNING   ****7.86 OVERDRAFT**

                255 00    247.14
```

Problem 10.4

```
 10 REM *POLYGONS
 20 RAD
 30 PRINT " n";TAB(9);"P",TAB(24
    ),"P/d"
 40 PRINT
 50 N=3
 60 D=35
 70 P=D*N*SIN(PI/N)
 80 PRINT N;TAB(2);P,TAB(18),P/D
 90 N=N+1
100 GOTO 70
110 END
```

Printer:

```
n       P                  P/d

3
 90.9326673975     2.59807621136
4
 98.9949493662     2.82842712475
5
 102.862419151     2.93892626146
6
 105               3
7
 106.301516084     3.03718617383
8
 107.151361062     3.06146745891
9
 107.736345148     3.07818128994
10
 108.155948031     3.09016994374
11
 108.467034384     3.09905812526
12
 108.703998943     3.10582854123
13
 108.888627251     3.11110363574
14
 109.035257638     3.11529307537
15
 109.153637679     3.11867536226
```

Problem 10.5

```
 10 REM *WEIGHT PLOT
 20 DATA 3.2,9.5,11.3,13.9,15.7,
    17.6,19.1,21.9,24.3,28.1,32.
    4,37.1,41.5,46.2,50.5
 30 DATA 53.8,55.7,56.7,56.7
 40 PRINT USING 50 ; 0,10,20,30,
    40;50
 50 IMAGE 2X,D,X,5(3X,2D)
 60 PRINT USING 70
 70 IMAGE "  +",5("-----+")
 80 FOR I=0 TO 18
 90 READ W
100 PRINT VAL$(I);TAB(3);"+";TAB
    (3+W/2),"*"
110 NEXT I
120 END
```

Printer:

```
  0     10    20    30    40    50
  +-----+-----+-----+-----+-----+
0 +  *
1 +     *
2 +       *
3 +        *
4 +         *
5 +          *
6 +           *
7 +             *
8 +              *
9 +                *
10+                   *
11+                      *
12+                         *
13+                           *
14+                              *
15+                                *
16+                                 *
17+                                 *
18+                                 *
```

Section 12

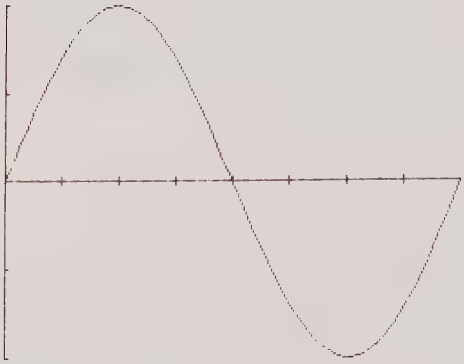
Problem 12.1

```
 10 REM *CARDIOID
 20 PEN 1 @ GCLEAR
 30 SCALE -3,1,-2,2
 40 RAD
 50 FOR T=0 TO 2*PI STEP PI/25
 60 MOVE 0,0
 70 R=1-COS(T)
 80 DRAW R*COS(T),R*SIN(T)
 90 NEXT T
100 END
```

Problem 12.2

```
 10 REM *PAD SINE CURVE
 20 GCLEAR
 30 SCALE 0,2*PI,-1,1
 40 XAXIS 0,PI/4
 50 YAXIS 0,.5
 60 RAD
 70 MOVE 0,0
 80 FOR X=0 TO 2*PI+.3 STEP PI/2
    0
 90 DRAW X,SIN(X)
100 NEXT X
110 END
```

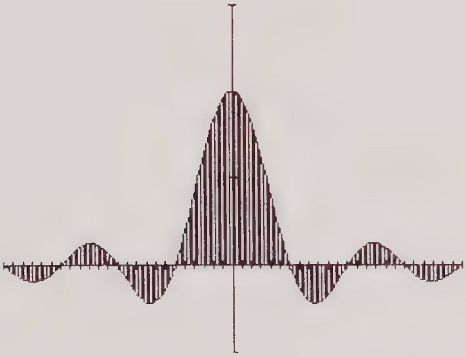
Display:



Problem 12.3

```
 10 REM *FILL SIN(X)/X
 20 GCLEAR
 30 RAD
 40 SCALE -4*PI,4*PI,-.5,1.5
 50 YAXIS 0.,5
 60 XAXIS 0,PI/6
 70 FOR X=-4*PI TO 4*PI STEP PI/
    20
 80 IF X=-4*PI THEN MOVE X,SIN(X
    )/X
 90 IF X=0 THEN 130
100 DRAW X,SIN(X)/X
110 MOVE X,0
120 DRAW X,SIN(X)/X
130 NEXT X
140 END
```

Display:



Problem 12.4

Run two programs back to back. The `SCALE` statement in the first program would be:

```
60 SCALE -36000,0,-36000,36000
```

The `SCALE` statement in the second program would be:

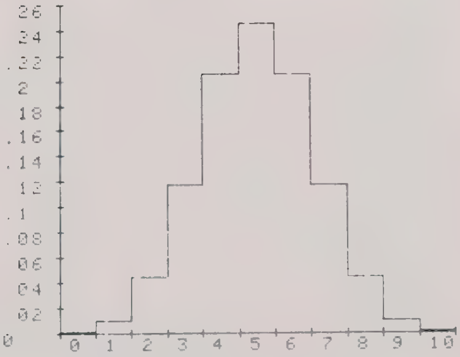
```
60 SCALE 0,36000,-36000,36000
```

Before you run the second program, copy the graphics screen onto the printer. When you copy the graphics from the second program, without advancing the paper, you will have a spiral twice as wide as the original design.

Problem 12.5

```
 10 REM *DISTRIBUTION OF HEADS*
 20 GCLEAR @ PEN 1
 30 SCALE -1.5,11,-.015, 26
 40 XAXIS 0,1,0,11
 50 YAXIS 0,.02,0, 26
 60 REM *LABEL X-AXIS*
 70 LDIR 0
 80 FOR X=0 TO 10
 90 MOVE X+.3,-.015
100 LABEL VAL$(X)
110 NEXT X
120 REM *LABEL Y-AXIS*
130 FOR Y=0 TO .26 STEP .02
140 MOVE -1.5,Y-.01
150 LABEL VAL$(Y)
160 NEXT Y
170 REM *PLOT HISTOGRAM*
180 N=10
190 P,Q=.5
200 PRINT "# HEADS","PROBABILITY
    "
210 MOVE 0,0
220 FOR R=0 TO 10
230 D=P^(N-R)*Q^R*FNF(N)/(FNF(R)
    *FNF(N-R))
240 PRINT R,INT(D*1000+ 5)/1000
250 DRAW R,D
260 IDRAW 1,0
270 NEXT R
280 DRAW 11,0
290 DEF FNF(X)
300 F=1
310 FOR I=X TO 1 STEP -1
320 LET F=F*I
330 NEXT I
340 FNF=F
350 FN END
360 END
```

Display:



Printer:

```
# HEADS              PROBABILITY
 0                      .001
 1                      .01
 2                      .044
 3                      .117
 4                      .205
 5                      .246
 6                      .205
 7                      .117
 8                      .044
 9                      .01
 10                     .001
```

Problem 12.6

```
 10 REM *HANGMAN
 20 DIM X(10),D$[25],R$[25],A$[2
    8],Q$[52],C$[1],G$[1],B$[26]
 30 INTEGER Y(140)
 40 Q$="abcdefghijklmnopqrstuvwx
    yzABCDEFGHIJKLMNOPQRSTUVWXYZ
    "
 50 I1=1
 60 GCLEAR @ CLEAR
 61 SCALE -5,35,-10,20
 62 DEG
 70 DISP "HAVE A FRIEND ENTER A
    WORD--AT MOST 23 CHARACTERS,
    SPACES NOT ALLOWED--NO PEEKI
    NG!!"
 80 DISP "WORD";
 90 INPUT A$
100 IF POS(A$,"")#0 THEN 80
101 CLEAR ! CLEAR ALPHA DISPLAY
110 GOSUB 1000 ! DRAW SCAFFOLD
120 L=LEN(A$)
130 L1=12-INT(L/2)*1.25
140 R$[1,25]=""
150 D$[1,25]=""
160 B$[1,26]=""
170 R$=A$
180 C=0
190 J=0
200 D$[1,L]="-------------------
    ---------------------------
    --"
210 MOVE L1,-2
220 LABEL D$
230 MOVE -5,-4 @ GCLEAR -2
240 LABEL "WHAT IS YOUR GUESS?"
250 MOVE -5,-6 @ GCLEAR -4
260 LABEL "YOU HAVE "&VAL$(6-C)&
    " GUESSES LEFT" @ MOVE -5,-8
270 C$="-"
280 INPUT C$
310 X=POS(Q$,C$)
320 IF X=0 THEN 250
330 IF X<26 THEN 360
340 X=X+26
350 C$=Q$[X,X]
360 X=POS(B$,C$)
370 IF X=0 THEN 530
380 DISP "YOU TURKEY! YOU ALREAD
    Y GUESSED THAT! NOW TRY AGAI
    N "
390 BEEP
400 WAIT 4000
410 GOTO 250
530 B$[J+C+1]=C$
540 X=POS(R$,C$)
550 IF X=0 THEN 810
560 N=1
570 J=J+1
575 ! CHECK TO SEE IF SAME LETTE
    R OCCURS MORE THAN ONCE
580 D$[X,X]=C$
590 R$[X,X]=" "
595 ! BLANK OUT OCCURENCES OF LE
    TTERS TO CHECK FOR MORE
600 X(N)=X
610 X=POS(R$,C$)
620 IF X=0 THEN 670
640 B$[J+C+1]=C$
650 N=N+1
660 GOTO 570
670 FOR I=1 TO N
680 MOVE L1,-1
685 FOR I4=1 TO X(I)-1
690 IMOVE 1.27,0
700 NEXT I4
705 LABEL C$
710 ! RESTORE WORD TO ORIGINAL F
    ORM
720 R$[X(I),X(I)]=C$
730 NEXT I
740 IF J<L THEN 250
750 BEEP
```

(Continued on next page.)

Problem 12.6 (Cont)

```
760 DISP D$;"  AW SHUCKS! YOU WI
    N!"
765 WAIT 1000
770 GRAPH
780 WAIT 3000
790 GOTO 880
810 DISP C$;" --WRONG!!!!!!"
820 WAIT 400
830 C=C+1
840 GOSUB 2000
850 IF C<6 THEN 250
860 BEEP
865 PRINT "HA HA!! I WIN!!!"
870 PRINT "BY THE WAY, THE WORD
    WAS ";R$
880 DISP "DO YOU WANT TO PLAY AG
    AIN (Y/N)?"
890 INPUT C$
900 IF UPC$(C$)="Y" THEN 50
910 IF UPC$(C$)="N" THEN 930
920 GOTO 900
930 GRAPH
940 END
1000 ! DRAW SCAFFOLD
1010 GCLEAR
1020 PENUP
1030 MOVE 8,1
1040 IDRAW -1,0 @ IDRAW 0,18
1050 IDRAW 7,0 @ IDRAW 0,-3 @ ID
     RAW 0,2
1060 IDRAW -6,0 @ IDRAW 0,-17
1070 IMOVE 0,14 @ IDRAW 3,3
1080 IMOVE 1,0 @ IDRAW -4,-4
1090 RETURN
2000 ON C GOTO 2010,2400,2600,28
     00,3000,3200
2010 ! DRAW THE HEAD
2020 Y=COS(45)
2030 X=1-Y
2040 MOVE 14,15
2050 FOR I=1 TO 2
2060 IDRAW -Y,-X
2070 IDRAW -X,-Y
2080 IDRAW X,-Y
2090 IDRAW Y,-X
2100 X=-X @ Y=-Y
2110 NEXT I
2120 IMOVE 0,-2
2130 IDRAW 0,-.2
2140 GOTO 4000
2400 ! DRAW THE LEFT ARM
2410 MOVE 14,12.5
2420 IDRAW -3,-2.5
2430 GOTO 4000
2600 ! DRAW THE RIGHT ARM
2610 MOVE 14,12 5
2620 IDRAW 3,-2.5
2630 GOTO 4000
2800 ! DRAW THE TRUNK
2810 MOVE 14,12.5
2820 IDRAW 0,-4.5
2830 GOTO 4000
3000 ! DRAW THE LEFT LEG
3010 MOVE 14,8
3020 IDRAW -3,-5
3030 IDRAW -1,0
3040 GOTO 4000
3200 ! DRAW THE RIGHT LEG
3210 MOVE 14,8
3220 IDRAW 3,-5
3230 IDRAW 1,0
4000 RETURN
```

Problem 12.7

```
10 REM *CREATE LANDSCAPE
1000 PEN 1 @ GCLEAR
1010 DEG
1020 SCALE 0,255,0,191
1030 REM *DRAW MOUNTAINS
1040 PENUP
1050 FOR A=-15 TO 255 STEP 15
1060 PLOT A+15,50*(COS(180/255*A
)+1)
1070 NEXT A
1080 PENUP
1090 FOR B=106 TO 255 STEP 15
1100 PLOT B,100*SIN(90/255*B)+10
```

Problem 12.7 (Cont)

```
1110 NEXT B
1120 REM *DRAW LAKE
1130 XAXIS 10,0,0,190
1140 H=100/TAN(10)+10
1150 R=100/SIN(10)
1160 PENUP
1170 FOR C=170 TO 190 STEP 1
1180 PLOT R*SIN(C)+90,R*COS(C)+H
1190 NEXT C
1200 REM *PLOT TREES
1210 T$=CHR$(8)&CHR$(28)&CHR$(62
     )&CHR$(127)
1220 FOR X=4 TO 240 STEP 60
1230 MOVE X,25
1240 GOSUB 2000
1250 NEXT X
1260 FOR X=52 TO 200 STEP 40
1270 MOVE X,29
1280 GOSUB 3000
1290 NEXT X
1300 MOVE 15,35
1310 GOSUB 3000
1320 MOVE 100,84
1330 GOSUB 2000
1340 MOVE 4,116
1350 GOSUB 3000
1360 MOVE 190,15
1370 GOSUB 3000
1380 MOVE 248,124
1390 GOSUB 2000
1400 MOVE 240,124
1410 GOSUB 3000
1420 MOVE 220,50
1430 GOSUB 3000
1440 MOVE 200,60
1450 GOSUB 3000
1460 REM *DRAW CLOUDS
1470 XAXIS 190,0,150,250
1480 XAXIS 185,0,140,253
1490 XAXIS 180,0,130,255
1500 XAXIS 175,0,135,250
1510 XAXIS 170,0,145,245
1520 XAXIS 165,0,150,225
1530 GOTO 5000
2000 FOR I=1 TO 3
2010 BPLOT T$,1
2020 NEXT I
2030 RETURN
3000 FOR I=1 TO 4
3010 BPLOT T$,1
3020 NEXT I
3030 RETURN
5000 REM *BPLOT MAN IN MOON*
5010 DIM M$[46],M2$[69]
5020 MOVE 0,191
5030 GOSUB 5140
5040 BPLOT M$,2
5050 REM *MOVE MOON
5060 FOR Y=191 TO 0 STEP -1
5070 FOR X=0 TO 255 STEP 8
5080 WAIT 1000
5090 MOVE X,Y
5100 BPLOT M2$,3
5110 MOVE X,Y
5120 NEXT X
5130 NEXT Y
5140 M$="E◀⊢◀?@▒E‡xπ1πΣΓατα▒▒⊢ĿĿ
     ĿSα‡_▒_◀_◀αᵒ1?x?E⊢`⊢◀E◀"
5150 M2$="EE◀⊢⊢◀?Ŀ@▒αE‡wxπα1πyΣΓ
     1αταα▒◀▒⊢◀ĿĿ◀ĿS<α‡E_▒@_◀__◀
     ααᵒ±1?Gx?DE⊢▒`⊢⊢◀EE◀"
5160 RETURN
5170 END
```

Index

Bold page numbers denote primary references, regular page numbers denote secondary references.

A

- ABS (absolute value), 59, **60**
- Accessories, **265-268**
- ACS (arccosine), **66**
- Active keys, **293**
- Adding statements, **96**
- Addition (+), **43**
- Advanced plotting (see BPLOT), **237-253**
- Allocating memory to program variables, **99**
- ALPHA, **197**
- Alpha display, **225**
- Ampersand (&), **52**
- AND operator, **55**
- Antilogarithm, **65**
- Area of circle program, **111**
- Argument, **59**
- Arithmetic, **43**
- Arithmetic hierarchy, **45**
- Array concepts, **119-120**
  - Lower bond, 119-120, **121**
  - One-dimensional, **119**
  - Subscripts, **119**
  - Two-dimensional, 119, **120**
- Array elements, **120**
  - Assigning values to, **133-135**
  - Tracing, **255-256**
- Array variables, **49**
  - Storing and retrieving, **191**
- Arrow keys (cursor positioning), **19**, 38
- ASN (arcsine), **66**
- ASSIGN#, **183-184**
- Assigning values to program variables, 90-91, **133-140**
  - From the keyboard with INPUT, **87-88**
  - With [LET], **90-91**
  - With READ and DATA, **137-140**
- Assigning values to variable names, **50**
- Assignments, **90-91**
- At symbol (@), **92**
- ATN (arctangent), **66**
- ATN2 (arctangent of x,y coordinate position), 66, **67-69**
- AUTO (automatic statement numbering), **28, 80**
- Auto key, **28, 80**
- Autost (autostart program), 18, **180**
- Averaging program, **28**
- Axes drawing programs, **202-206**

B

- Backspace, **38**
  - Fast, **38**
- Backspace with graphics mode input, **230**, 231, 236
- Backwards spelling program, **129**
- Base conversions program, **257**
- BASIC language, 76-77, **295-301**
  - Predefined functions, summary, **300-301**
  - Statements, syntax summary, **295-301**
  - Syntax guidelines, **78-79**, 297
- BASIC programming, introduction, **76-83**
- BASIC typewriter mode, **33**
- BEEP, **89-90**
  - Chime program, **156**
  - Key of C Major program, **155**
- Binary programs, **193**
- Blank spaces, **162**
- BPLOT (byte plot), **237-253**
  - Condensing the string assignment program, **243**
  - Man in the moon example, **244-252**
  - Moving a figure on the graphics display, **247-250**
  - Procedure for building the string, **238-240**
  - Using the string with BPLOT, **241-242**
- Brackets, 45, 122, **124-125**
- Branching, **103-117, 145-159**
  - Computed GOSUB, **153**
  - Computed GOTO, **108-110**
  - Conditional (IF...THEN...ELSE), **103-108**
  - Defining functions, **145-151**
  - FOR-NEXT loops, **110-116**
  - Special function keys, **154-156**
  - Subroutines, **151-153**
  - Unconditional (GOTO), **91, 108-110**
- Brightness, display, **36**, 273
- Buffer, **183-184**
- Buffer number, **183**, 184, 185
- Bytes, 97, **141**, 142, 176, 181-183

C

- Calculator mode, **18**, 43, 50
- Calendar Functions program, **23**
- Cancelling key assignments, **156**
- Capital letters, **33**
- Caps lock key, **33-34**
- Cardioid program, **214-215**
- Carriage return and line feed, **161-162**
- CAT (catalog), 26, **175-176**
- Cathode ray tube (see also, CRT), **36**
- CEIL (ceiling), 59, **61**
- Celsius/Fahrenheit conversion program, **106**
- CHAIN, **179-180**
- Character codes, **292**
- Character conversions, **131-133**
  - CHR$, **131**
  - NUM, **132**
  - UPC$, **133**
- Character set, **34**
- Character strings, **49**
- Checkbook balancing program, **79**
- Checking a halted program, **260**
- CHR$ (character), 34, 128, **131-132**
- Circle approximation program, **212**
- Circle program, **200-201**
- Class program, **186**
- Cleaning, general, **285**
- CLEAR, **19**, 77, 298
- Clear key, **19**, 37, 98, 230
- Clear to end of line, **19**
- Clearing computer memory, **78**
- Clearing the display, **19**
- Clearing the graphics display, **199**
- Clock label program, **222**
- Closing a data file, **184**
- COM (common), 121, **123**
- Comma, 47, **84**, 161
  - Insert commas program, **150**
  - Replace decimal point with comma program, **149**
- Commands, 77-78, **298**
  - Non-programmable, **78**
  - Programmable, **78**
  - Syntax summary, **297**
- Common antilogarithm, **66**
- Common logarithm, **65**

Compacted field specifier, K (IMAGE), **166**
Computed GOSUB, **153**
Computed GOTO, **108-110**
Concatenation (&—string), **52**
Conditional branching, **103-108**
Conditioning the tape, **282**
Conserving memory, 141-142, **263**
CONT (continue), **98, 99-100**
Cont key, 26, **98**, 99-100
Control characters, **34**, 292
Control (ctrl) key, **34**
COPY, **35**, 198
Copy key, **35**, 98, 198, 230
COS (cosine), **66**
COT (cotangent), **66**
CREATE, **180**
Creating a data file, **180-183**
Creating a program, 27, **78-79**
CRT (cathode ray tube) display, 10, **36**
  Brightness control knob, **271**
  Service, **286**
CRT IS, **169-170**
CSC (cosecant), **66**
CTAPE (condition tape), **282**
Ctrl (control) key, **34**
Cursor, **17**, 38
Cursor positioning keys, **19**, 38
Curves, **212-215**

D

DATA (with READ), **137-139**
Data cartridges (see also tape, tape cartridge), **30**
Data file (see also, file), **180-190**
Data precision, **49**, 296
Data storage (tape), **181-183**
DATE, **57**
Debugging and error recovery, **255-263**
Decimal character codes, 34, **292**
Decimal to octal conversion program, **147-148**
Decision, **103-108**
Declaratory statements, **76-77**, 105
Declaring and dimensioning variables, **121-124**
DEF FN (define function), **145**, 147
Default error processing, **69-70**, 101
DEFAULT OFF, **70**, 101
DEFAULT ON, **70**, 101
Default values for math errors, **69**, 303
DEG (degrees), **66**
Degrees/radians conversions, **67**
Del (delete) key, **95**
Delaying program execution, **100**
DELETE, **95**
Delete character key, **38**
Delete line key, **20**, 38
Deleting program statements, **95-96**
Delimiters, **161**
Digit separator symbols (IMAGE), **165**
Digit symbols (IMAGE), **163**
DIM (dimension), **121-122**
Directory, tape, **175**
DISP (display), **84-86**
DISP USING, **161, 167-168**
Display, **36-39**
  Brightness control, **36**, 271
  Editing, **38-39**
  Formatting, **161-173**
  Graphics, **197-198**, 225
  Service, **286**
DIV (integer division), **44**
Division (/), **43**
DO IF TRUE rule, **104**
Dot matrix, 78, **227**, 297
DRAW, **211**
Drawing coordinate axes, **202-206**
Drawing curves, **212-215**
DTR (degrees to radians), 66, **67**

E

E (exponent), **48**, 165
e^x, **65**
Editing keyboard lines, **38-39**
Editing programs, **95-101**
Eject bar, **23**
ELSE, **106-108**
END, 27, **77**
End line key, 29, **81**
End of file and record marks, **188-189**, 190
  Errors, **183**, 186, 187, 189, 190
Entering long expressions, **36**
Entering program statements into computer memory, **81**
Entering a program, 27, **79-82**
EPS (epsilon), 28, 61, **64**
Equal to (=), **53**
ERASETAPE, 30, **175**
Erasing a program from the tape cartridge, 31, **192**
ERRL (error line), **261**
ERRN (error number), **261**
Error messages, 20, **100-101**, 303-307
  Summary, **303-307**
Error recovery functions, **261**
Error testing and recovery, **260-263**
Executable statements, **76-77**, 105
EXOR operator, **55**, 241, 247
EXP (e^x, natural antilogarithm), **65**
Exponent, **48**
Exponent symbol (IMAGE), **165**
Exponentiation (∧), **43**, 44
Exponents of ten, **48**
Expressions, **43**
Expressions and keyboard operations, **43-57**

F

Factorial program, **114**
Fast backspace, **38**
File, tape, **176**, 180-190, 192, 193-195
  Closing a, **184**
  Creating a, **180**
  Data, **180-190**
  Name, **180**
  NULL, **192**
  Number, **176**
  Opening a, **183-184**
  Pointer, **185**, 190
  Program, **176**
  Random access, **188-190**
  Securing a, **193-195**
  Serial access, **185-188**
  Type, **176**
FLIP, **34**
FLOOR, 59, **61**
FN, **147**, 299
FN END (function end), **147**
FOR (with NEXT), **111**
FOR-NEXT loops, **110-116**, 213-215
  Considerations, **116**
  Padding the increment, **213-215**
  STEP, 111, **114-15**
Format of numbers, **47**
Formatted output, 84-87, **161-173**
  TAB function, **168-169**
  With IMAGE, **161-167**
  Within PRINT/DISP USING statements, **167-168**
FP (fractional part), 59, **60**
Frame graphics display program, **197**
Function keys, special, **154-156**
Functions, **59**
  Error, **261**
  Math, **59-70**
  Summary, **300**
  Multiple-line, **147-151**

  - Single-line, **145-146**
  - String, **128-133**
  - User-defined, **145-151**
- Fuse, **270**
  - Installation, **272**
  - Receptacle, **271**
- Future value program, **99**

G

- GCLEAR (graphics clear), **199**
- Getting started, **17-31**
- Glossary, **295-297**
- GOSUB, **151**
  - Computed, **153**
- GOTO, **91**
  - Computed, **108-110**
- GRAD (grads mode), **66**
- GRAPH, **197**
- Graph key, 98, **197**, 230
- Graphics, **197-253**
  - Display, **197-198**, 225
  - Input in graphics mode, **230-237**
  - Printer and, **198**
  - Statements, syntax summary, **300**
- Greater than (>), **53**
- Greater than or equal to (>=), **53**
- Greatest integer function, **60-61**
- Ground information, **271**
- Grounding requirements, **270**

H

- Halted program, checking a, **260**
- Halting program execution, 26, **77**, 97-98
- Hierarchy, arithmetic, **45**
- Hierarchy, math, **69**
- Home key, **19**
- Home position, **17**
- Hypotenuse program, **27**

I

- IDRAW (incremental draw), **217-220**
- IF...THEN, **103-108**
- IF...THEN...ELSE, **106-108**
- IMAGE (with PRINT/DISP USING), **161-168**
  - Format string, **161**
  - Field overflow, **167**
  - Reusing the format string, **166**
  - Replication, **166**
  - Summary table, **168**
- IMOVE (incremental move), **217**
- INF (infinity), 61, **64**
- INIT (initialize), **99**
- Init key, **99**
- Initial set-up instructions, **271-273**
- Initializing a program, **99**
- Initializing variables, **136**
- INPUT, **87-88**
- Input in graphics mode, **230-237**
- Input prompt (?), **87**, 232, 236
- Insert commas program, **150**
- Insert mode, **39**
- Insert/replace key, **39**
- Inserting characters, **39**
- Inspection procedure, **269-271**
- Ins/rpl (insert/replace) key, **39**
- INT (greatest integer), 59, **61**
- INTEGER, 49, 121, **122-123**
- Integer division (DIV or \\), **44**
- Interrupting program execution, **97-100**
- IP (integer part), 59, **60**

K

- Key codes, **293**
- Key index, **12-13**
- KEY LABEL, **154**
- Key label key, 98, **154**, 230
- Key labels, examples, **23**, **154**, **155**
- Key number, **154**
- Key of C Major program, **155**
- Key response during program execution, **293**
  - In ALPHA mode, **97-98**, 230, 293
  - In GRAPH mode, **230**, 293
- Keyboard, **33**
- Keyboard arithmetic, **43**
- Keyboard, printer, and display control, **33-40**
- Keying in exponents of ten, **48**
- Keys, special function, **154-156**
- Keywords, 33, 34, **76-77**

L

- LABEL, **221**
- Labeling graphs, **221-230**
  - Label direction, **224**
  - Label length, **225**
  - Label positioning, **227-230**
- Language (BASIC), 7, 76-77, **298-301**
- LDIR (label direction), **224**
- LEN (length), **128-129**
- Length of a string variable, **51**
- Length of statement, **36**
- Length of expression, **36**
- Less than (<), **53**
- Less than or equal to (<=), **53**
- LET, **90-91**
- LET FN, **147**, 299
- LGT (log to base ten), **65**
- Line generation, **198**
- Line numbers (see also, statement numbers), **77**
- Line voltage selector switch, **271**
- LIST, **97**
- List key, **75**
- Lists, **119**
- Literal strings, **124**
- Live keys, **98**, 293
- LOAD, 23, 74, **179**
- LOAD BIN, **193**
- Load key, 23, **74**
- Loading a program from the Standard Pac, **22**
- Loading a prerecorded program, **74**, 179
- LOG (natural logarithm), **65**
- Logarithmic functions, **65-66**
- Logical evaluation, **53-56**
- Logical operations, **54-56**
- Logical record, **181**
- Loops, 91, **110-116**
  - Loop counter, **111**
  - Nested, **115**
- Lower bounds of arrays, **121**

M

- Man in the moon BPLOT program, **249-252**
- Manual problem solving, **18**
- Math hierarchy, **69**
- Mathematics functions and statements, **59-70**
- Matrices, **119**
- MAX (maximum), 61, **62**
- Memory, 97, **140-142**, 263
  - Conserving, 141-142, **263**
  - Read/write, **140**, 141
  - Remaining, **97**
  - System, **140-142**
- MIN (minimum), 61, **62**
- Minus sign (−), **43**
- MOD (modulo), **44**

Modifying string variables, **125-127**
Module installation and removal, general, **273-275**
Module plug-in ports, **271**
Modulo, **44**
MOVE, **211**
Multiple-line functions, **147-151**
Multiplication (*), **43**, 44
Multistatement lines, **92**

N

Name of tape file, **176**
Natural antilogarithm, **65**
Natural logarithm, **65**
Nested loops, **115**
NEXT, **111**
Non-programmable commands, **78**
Normal typewriting mode, **34**
NORMAL, 22, **35**, **80**, 256
Not equal to (<> or #), **53**
NOT operator, **55**
NULL file, 182, **192**
Null string (“”), **52**
NUM (numeric), 128, **132**
Number alteration, **59**
Number base conversions program, **257**
Number entry, **43**
Numbers, range of, **49**
Numbers, standard format, **47**
Numeric expression, 43, **297**
Numeric specification (with IMAGE), **163-165**
Numeric variables, **49**

O

OFF ERROR, **261**
OFF KEY#, **156**
OFF TIMER#, **157**
ON ERROR, **261**
ON KEY#, **154**
ON...GOSUB, **153**
ON...GOTO, **108-110**
ON TIMER#, **156**
ON/OFF switch, **271**
Opening a data file, **183-184**
Operators, **43**
Operators, summary, **295**
OPTION BASE, **121**
OR operator, **55**
Order of execution
    Expressions, **46**
    Math operators, **69**
    Program, **83**
Output codes, **169**
Owner’s information, **269-289**

P

Padding a FOR-NEXT loop, **213-215**
Paper, loading printer, **277-279**
Paper advance key, **35**, 98, 230
Parentheses, **45**
Payroll program, **109**
PAUSE, **99-100**
Pause key, 26, 75, **97-98**
PEN, **207**
PENUP, **207**
Physical record, **181**
PI, 61, **63**
PLIST (printer list), **97**
Plist key, **75**
PLOT, **208**
Plot figure without lifting pen program, **209**
Plot “twinkling” star program, **209**
Plot star clusters program, **210**
Plotting operations, **207-211**
Plug-in module installation, **273-276**
Plug-in ROM, **275**
Plus sign (+), **43**
Pointer, repositioning, **190**
Polar/rectangular coordinate conversions, **67-69**
POS (position), 128, **129**
Positioning labels, **227-230**
Power cords, **269-270**
Power on, **17**
Power requirements, **270**
Power supply, **269-270**
Power switch, **17**
Powers, **43**, 44
Precision (accuracy), **49**, 296
Prerecorded programs, 73, **74**
PRINT, **86-87**
PRINT ALL, 21 **35**, 37
PRINT and DISP, **46**
Print mode, **35**
PRINT USING, **161**, **167-168**
PRINT#, **185**, **188**
Printer, 35, **276-279**
    Access, **35**
    Formatting, **161-173**
    Intensity dial, **35**
    Maintenance, **279**
    Paper, **277**
    Paper loading, **277-279**
    Service, **287**
PRINTER IS, **169-170**
Problems, examples at end of sections, **93**, **101**, **116-117**, **142-143**, **158-159**, **170-173**, **216-217**, **220**, **230**, **237**, **253**
Problems, sample solutions, **309-336**
Program editing, **95-101**
Program name, **176**
Programmable commands, **78**
Programs, storing on tape, **176-178**
Programs, retrieving programs form tape, **179**
Protecting (securing) tape files, **193**
Protecting a tape cartridge, **30**
PURGE, 31, **192**

Q

Quadratic roots program, **107**
Question mark (?), **87**
Quotation marks, 27, 52, 88, 110, **124**, **162**, 167-168

R

RAD (radians), **66**
Radix symbols (IMAGE), **164**
Random data plot program, **231**
Random file access, **188-190**
Random number seed, **64**
Random numbers, **64-65**
Random reading, **190**
Random writing, **188**
RANDOMIZE, **64-65**
Range of numbers, **49**
Range, computing, **47**
Range, storage, **49**
READ (with DATA), **137-139**
Read only memory (ROM), **140**
READ#, **187**, **190**
Read/write memory (RAM), **140**, 141
REAL, 49, 121, **122-123**
Rear panel, **10**, 271
Reciprocal program, **105**
RECORD slide tab, 22, 30, **281**
Recording a program, 30, **176-178**
Records, **180-181**

Records (Cont)
    Logical, **181**
    Physical, **181**
Recovering from math errors, **69-70**
RECS (records), **176**
Rectangular/polar coordinate conversions, **67-69**
Redefining the printer and the display, **169-170**
Reference tables, **291-293**
Relational operations, **53-54**
REM (remark), **83**
Remainder function (RMD), **62-63**
REN (renumber), **96**
RENAME, **192**
Repair policy, **288**
Replace decimal point with comma program, **149**
Replace mode, **39**
Replication (IMAGE), **166**
Rereading data, **139-140**
Reset conditions, **291**
Reset key, **40**
Resetting the computer, **40**
RESTORE (with READ, DATA), **139-140**
Result key, **46**
Resuming program execution, **98**
Retrieving a program from tape, **179**
RETURN (with GOSUB), **151**, 152, 153
Reverse letter order program, **129**
REWIND, **280**
RMD (remainder), 61, **62-63**
RND (random number), 61, **64-65**
Roll key, **36**, 98, 230
ROM drawer, **275**
ROM installation and removal, **275-276**
Roots of quadratic equation program, **107**
Round to two decimal places program, **146**
RTD (radians to degrees), 66, **67**
RUN, 77, **99**
Run key, 23, **82**
Running a program, **27**
Running a prerecorded program, **22**
Run-time errors, **101**

S

Sales and service offices, **343-344**
Sample problems (see Problems)
Sample solutions, **309-336**
SCALE, **199**
Scaling the graphics display, **199-202**
    Equal unit scaling, **201**
    Unequal unit scaling, **200**
Scientific notation, **48**
Scores and averages program, **191**
SCRATCH, 27, 77, **78**
Scratch key, 77, **78**
SEC (secant), **66**
SECURE, **193-195**
Secure type, **194**
Security code, **193**, 194
Seed, random number, **64**
Select codes, **169**
Self-test, **39**
Self-test error, **40**
Semantic errors, **101**
Semicolons, 47, **84**
Serial file access, **185-188**
Serial number, **289**
Serial number plate, **271**
Serial reading, **187-188**
Serial writing, **185-187**
Service, **286-287**
Set-up instructions, **271-273**
SETTIME, **56**
SGN (sign), 61, **62**
Shift key, **33**
Shifted characters, **34**
Shipping instructions, **288**
SHORT, 49, 121, **122-123**
Sign of a number (SGN), **62**
Sign symbols (IMAGE), **164-165**
Simple display editing, **19**
Simple programming, **73-93**
Simple variables, 49, **50-51**
SIN (sine), **66**
Single-line functions, **145-146**
Ski game, **74**
Slash (/), **161**
Small letters, **34**
Smallest integer function, **61**
Solutions to example problems, **309-336**
Space bar, **38**
Spaces, **162**
Spacing, **18**
Spacing of program statements, **80**
Special characters, **296**
Special function keys, 23, **154-156**
Sort program, **177**
SQR (square root), 61, **62**
Standard number format, **47**
Standard Pac, **22**
Star clusters program, **210**
Statement, **76**
Statement length, **81**
Statement numbers, **77**
    Automatic numbering (AUTO), **80**
    Renumbering (REN), **96**
Statements, syntax summary, **295-301**
STEP increment value, 111, **114-115**
Step key, **259**
STOP, 77
Stopping a running program, **75**
Storage (tape), data, **181-183**
STORE, **176-178**
STORE BIN, **193**
Store key, **30**
Storing and retrieving data, **185-191**
Storing entire arrays, **191**
Storing variables, **141**
String allocation in user-defined functions, 145, **151**
String conversions, **130-133**
    Characters to numbers (NUM), **132**
    Lowercase to uppercase (UPC$), **133**
    Numbers to strings (VAL$), **131**
    Numbers to characters (CHR$), **131**
    Strings to numbers (VAL), **130**
String concatenation, **52**
String comparisons, **54**
String expressions, **124-133**
String functions, **128-133**
    CHR$, 128, **131-132**
    LEN, **128-129**
    NUM, 128, **132**
    POS, 128, **129**
    VAL, 128, **130-131**
    VAL$, 128, **131**
    UPC$, 128, **133**
String modification, **125-127**
String specification (with IMAGE), **162-163**
String variable, **51**
    Length, **122**, 124
    Name, **51**
Subroutines, **151-153**
Subscripts, **119**, 121, 125
    Arrays, **119-120, 122**
    Strings, **124-125**
Substrings, **125**
Subtraction (−), **43**
Summer Olympic Swimming Records program, **219**, 228
Syntax conventions, **78**
Syntax errors, **100**
Syntax guidelines to commands and BASIC statements, **297**
System errors, 39-40, **303-307**
System hints, **262-263**
System self-test, **39**

T

TAB, **168-169**
Tables, **119**
TAN (tangent), **66**
Tape care, **281-282**
Tape cartridge, 30, **280-285**
    Inserting, **280**
    Removing, **281**
    Rethreading, **283-284**
    Specifications, **280**
Tape cartridges, using, **175-195**
Tape directory, **175**
Tape drive, service, **287**
Tape drive light, **23**
Tape eject bar, **23**
Tape file, **30**
Tape life, **282-283**
Tape record, **180**
Tape storage medium, **175-195**, 280-285
    Conditioning, **282**
    General information, **280**
    Initializing, **175**
    Optimizing use, **285**
    Write protection, **281**
Tax program, **104**
Temperature ranges, **286**
Temperature conversion program, **106**
Test key, **39**
THEN, **103**
Tic spacing, **202,** 204
TIME, **57**
Time functions, **56-57**
Timer number, **156**
Timers, **156-158,** 287
TRACE (trace program branches), **255**
    Cancelling trace operation, **256**
TRACE ALL, **256**
TRACE VAR (trace variables), **255-256**
Triangle BPLOT program, **241**
Trigonometric modes, **66**
Trigonometric functions and statements, **66-69**
Truth table, **56**
Twinkling star program, **209**
Type declarations, **122-123**
    INTEGER, **122**
    REAL, **122**
    SHORT, **122**
Type of tape file, **176**
Types of variables, **49**
Typewriter keys, **33**

U

Unconditional branching, **91, 108-110**
    GOTO, **91**
    ON...GOTO, **108-110**
    ON...GOSUB, **153**
UNSECURE, **194**
UPC$ (uppercase), 128, **133**
User-defined functions, **145-151**
    String allocation in, 145, **151**

V

VAL (numeric value), 128, **130-131**
VAL$ (character value), 128, **131**
Variable types, **49**
Variable forms, **49**
Variables, 21, **49**
    Array, **49**
    Arrays and strings, **119-143**
    Declaring and dimensioning, **121**
    Memory storage, **141**
    Numeric, 49-51, **121-122**
    Simple, 49, **50-51**
    String, **51**
    Summary, **297**
Vectors, **119**
Voltage selection, **271-272**

W-Z

WAIT, **100**
Warranty, **287-288**
Warranty information toll-free number, **288**
Widget program, **113**
Workspace, **286**
Write protection (tape), 30, **281**
Writing a BASIC program, **78**
XAXIS, **202**
X^2 program, **145**
YAXIS, **202**

HEWLETT
PACKARD

SALES & SERVICE OFFICES

AFRICA, ASIA, AUSTRALIA

AUSTRALIA

Hewlett-Packard Australia
Pty Ltd
31-41 Joseph Street
Blackburn Victoria 3130
P O Box 36
Doncaster East, Victoria 3109
Tel 896351
Telex 31-024
Cable HEWPARD Melbourne

Hewlett-Packard Australia
Pty Ltd
31 Bridge Street
Pymble
New South Wales 2073
Te 4496566
Telex 21561
Cable HEWPARD Sydney

Hewlett-Packard Australia
Pty Ltd
153 Greenhill Road
Parkside S A 5063
Tel 2725911
Telex 82536
Cable HEWPARD Adelaide

Hewlett Packard Australia
Pty Ltd
141 Stirling Highway
Nedlands W A 6009
Tel 3865455
Telex 93859
Cable HEWPARD Perth

Hewlett Packard Australia
Pty Ltd
121 Wollongong Street
Fyshwick A C T 2609
Tel 804244
Telex 62650
Cable HEWPARD Canberra

Hewlett Packard Australia
Pty Ltd
5th Floor
Teachers Union Building
495-499 Boundary Street
Spring Hill, Queensland 4000
Tel 2291544
Cable HEWPARD Brisbane

HONG KONG

Hewlett-Packard Hong Kong Ltd
Austin Centre Room 105
First Floor
21 Austin Road
Kowloon
Tel 3-697446

INDIA

Blue Star Ltd
Kasturi Buildings
Jamshedji Tata Rd
Bombay 400 020
Tel 29 50 21
Telex 011-2156
Cable BLUEFROST

Blue Star Ltd
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
Bombay 400 025
Tel 45 78 87
Telex 011-4093
Cable FROSTBLUE

Blue Star Ltd
Band Box House
Prabhadevi
Bombay 400 025
Tel 45 73 01
Telex 011-3751
Cable BLUESTAR

Blue Star Ltd
Bhavdeep
Stadium Road
Ahmedabad 380 014
Tel 42880
Telex 234
Cable BLUEFROST

Blue Star Ltd
7 Hare Street
P O Box 506
Calcutta 700 001
Tel 23-0131
Telex 021-7655
Cable BLUESTAR

Blue Star Ltd
Bhandari House
7th & 8th Floor
91 Nehru Place
New Delhi 110 024
Tel 634770 & 635166
Telex 031 2463
Cable BLUESTAR

Blue Star Ltd
Blue Star House
11/11A Magarath Road
Bangalore 560 025
Tel 55668
Telex 043-430
Cable BLUESTAR

Blue Star Ltd
Meeakshi Mandiram
xxx 1678 Mahatma Gandhi Rd
Cochin 682 016
Tel 32069,32161 32282
Telex 0885-514
Cable BLUESTAR

Blue Star Ltd
1-1-117/1
Sarojini Devi Road
Secunderabad 500 003
Tel 70126, 70127
Telex 015-459
Cable BLUEFROST

Blue Star Ltd
2/34 Kodambakkam High Road
Madras 600 034
Tel 82056
Telex 041-379
Cable BLUESTAR

INDONESIA

BERCA Indonesia P T
P O Box 496/Jkt
Jln Abdul Muis 62
Jakarta
Tel 349255 349886
Telex 46748 BERSIL IA
Cable BERSAL

BERCA Indonesia P T
P O Box 174/Sby
23 Jln Jimerto
Surabaya
Tel 42027
Cable BErcacon

ISRAEL

Electronics Engineering Div
of Motorola Israel Ltd
16 Kremenetski Street
P O Box 25016
Tel-Aviv
Tel 38973
Telex 33569 34164
Cable BASTEL Tel-Aviv

JAPAN

Yokogawa-Hewlett-Packard Ltd
Chuo Bldg , 4th Floor
4-20 Nishinakajima 5-chome
Yodogawa-ku, Osaka-shi
Osaka 532
Tel 06-304 6021
Telex 523-3624

Yokogawa-Hewlett-Packard Ltd
29-21 Takaido-Higashi 3-chome
Suginami-ku **Tokyo** 168
Tel 03-331-6111
Telex 232 2024 YHP-Tokyo
Cable YHPMARKET TOK 23 724

Yokogawa-Hewlett-Packard Ltd
Nakamo Building
24 Kami Sasajima-cho
Nakamura-ku **Nagoya** 450
Tel 052 571-5171

Yokogawa Hewlett-Packard Ltd
Tanigawa Building
2-24-1 Tsuruya-cho
Kanagawa-ku
Yokohama 221
Tel 045-312-1252
Telex 382 3204 YHP YOK

Yokogawa Hewlett-Packard Ltd
Mito Mitsui Building
105 1-chome San-no-maru
Mito Ibaragi 310
Tel 0292 25-7470

Yokogawa-Hewlett-Packard Ltd
Inoue Building
1348-3 Asahi-cho 1-chome
Atsugi Kanagawa 243
Te 0462-24-0452

Yokogawa-Hewlett-Packard Ltd
Kumagaya Asahi
Hachijuni Building
4th Floor
3-4 Tsukuba
Kumagaya Saitama 360
Tel 0485 24-6563

KOREA

Samsung Electronics Co Ltd
15th Floor Daeyongak Bldg
25-5 1-KA
Choong Moo-Ro Chung-Ku
Seoul
Tel (23) 6811 778 3401 2 3-4
Telex 22575

MALAYSIA

Hewlett-Packard Sales SDN BHD
Suite 2 21 2 22
Bangunan Angkasa Raya
Jalan Ampang
Kuala Lumpur
Tel 483680 485653

Protel Engineering
P O Box 1917
Lot 259 Satok Road
Kuching **Sarawak**
Tel 53544
Cable PROTELENG

NEW ZEALAND

Hewlett Packard (N Z) Ltd
4-12 Cruickshank Street
Kilbirnie Wellington 3
P O Box 9443
Courtney Place
Wellington
Tel 877 199
Cable HEWPACK Wellington

Hewlett Packard (NZ) Ltd
P O Box 26-189
169 Manakau Road
Epsom
Auckland
Tel 687-159
Cable Hewpack Auckland

Analytical-Medical Only
Medical Supplies N Z Ltd
Scientific Division
79 Carlton Gore Road Newmarket
P O Box 1234
Auckland
Tel 75-289
Cable DENTAL Auckland

Analytical Medical Only
Medical Supplies N Z Ltd
Norrie and Parumoana Streets
Porirua
Tel 75-098
Telex 3858

Analytical Medical Only
Medical Supplies N Z Ltd
P O Box 309
239 Stanmore Road
Christchurch
Tel 892-019
Cable DENTAL Christchurch

Analytical Medical Only
Medical Supplies N Z Ltd
303 Great King Street
P O Box 233
Dunedin
Tel 88 817
Cable DENTAL Dunedin

PAKISTAN

Mushko & Company Ltd
Oosman Chambers
Abdullah Haroon Road
Karachi-3
Tel 511027 512927
Telex 2894
Cable COOPERATOR Karachi

Mushko & Company Ltd
388 Satellite Town
Rawalpindi
Tel 41924
Cable FEMUS Rawalpindi

PHILIPPINES

The Online Advanced
Systems Corporation
Rico House
Amorsolo cor Herrera Str
Legaspi Village Makati
P O Box 1510
Metro **Manila**
Tel 85-35-81 85 34 91 85 32 2
Telex 3274 ONLINE

Electronics Specialists & Proponents
Inc (Esp Inc)
P O Box 2649 Manila
Federation Bldg
Muelle de Binondo
Manila
Tel 48-46-10 48-46 25 47-28 46
Telex 742-0287
Cable Espinc Manila

RHODESIA

Field Technical Sales
45 Kelvin Road North
P O Box 3458
Salisbury
Tel 705231 (5 lines)
Telex RH 4122

SINGAPORE

Hewlett Packard Singapore
(Pte) Ltd
1150 Depot Road
Alexandra P O Box 58
Singapore 4
Tel 270 2355
Telex HPSG RS 21486
Cable HEWPACK Singapore

SOUTH AFRICA

Hewlett Packard South Africa
(Pty) Ltd
Private Bag Wendywood
Sandton Transvaal 2144
Hewlett Packard Centre
Daphne Street Wendywood
Sandton 2144
Tel 802 1040 8
Telex 8 4782
Cable HEWPACK Johannesburg

Hewlett Packard South Africa
(Pty) Ltd
P O Box 120
Howard Place Cape Province 7450
Pine Park Centre Forest Drive
Pinelands Cape Province 7405
Te 53 7955 thru 9
Telex 57 0006

SRI LANKA

Metropolitan Agencies Ltd
209 9 Union Place
Colombo 2
Tel 35947
Telex 1377METROLTD CE
Cable METROLTD

TAIWAN

Hewlett Packard Far East Ltd
Taiwan Branch
Bank Tower 5th Floor
205 Tun Hua North Road
Taipei
Tel 751 0404
Cable Hewpack Taipei

Hewlett Packard Far East Ltd
Taiwan Branch
68 2 Chung Cheng 3rd Road
Kaohsiung
Te (07) 242318 Kaohsiung

Analytical Only
San Kwang Instruments Co Ltd
20 Yung Sui Road
Taipei
Tel 3615446 9 (4 lines)
Telex 22894 SANKWANG
Cable SANKWANG Taipei

THAILAND

UNIMESA Co Ltd
Elcom Research Building
2538 Sukumvit Ave
Bangchak, Bangkok
Tel 3932387 3930338
Cable UNIMESA Bangkok

Bankok Business Equip Ltd
5 5 6 Dejo Road
Bangkok
Tel 234 8670 3
Cable Busiquipt Bangkok

OTHER AREAS NOT LISTED CONTACT

Hewlett Packard Intercontinental
3495 Deer Creek Rd
Palo Alto California 94304
Tel (415) 856 1501
TWX 910 373 1267
Cable HEWPACK Palo Alto
Telex 034 8300 034 8493

CANADA

ALBERTA

Hewlett-Packard (Canada) Ltd
11620A - 168th Street
Edmonton T5M 3T9
Tel (403) 452-3670
TWX 610-831-2431

Hewlett-Packard (Canada) Ltd
210 7220 Fisher St S E
Calgary T2H 2H8
Tel (403) 253-2713
Twx 610-821-6141

BRITISH COLUMBIA

Hewlett-Packard (Canada) Ltd
10691 Shellbridge Way
Richmond V6X 2W7
Tel (604) 270-2277
TWX 610-925-5059

MANITOBA

Hewlett-Packard (Canada) Ltd
380-550 Century St
Winnipeg R3H 0Y1
Tel (204) 786-6701
TWX 610-671-3531

NOVA SCOTIA

Hewlett-Packard (Canada) Ltd
800 Windmill Road
Dartmouth B3B 1L1
Tel (902) 469-7820
TWX 610-271-4482

ONTARIO

Hewlett-Packard (Canada) Ltd
1020 Morrison Dr
Ottawa K2H 8K7
Tel (613) 820-6483
TWX 610-563-1636

Hewlett-Packard (Canada) Ltd
6877 Goreway Drive
Mississauga L4V 1M8
Tel (416) 678-9430
TWX 610-492-4246

Hewlett-Packard (Canada) Ltd
552 Newbold Street
London N6E 2S5
Tel (519) 686-9181

QUEBEC

Hewlett Packard (Canada) Ltd
275 Hymus Blvd
Pointe Claire H9R 1G7
Tel (514) 697-4232
TWX 610-422 3022
TLX 05-821-521 HPCL

FOR CANADIAN AREAS NOT LISTED

Contact Hewlett Packard (Canada)
Ltd in Mississauga

2/79

CENTRAL AND SOUTH AMERICA

ARGENTINA

Hewlett-Packard Argentina
S A
Av Leandro N Alem 822 - 12°
1001 **Buenos Aires**
Tel 31-6063 4 5,6
Telex 122443 AR CIGY
Cable HEWPACKARG

Biotron S A C I y M
Bolivar 177
1066 **Buenos Aires**
Tel 30-4846 34-9356 34-0460
33 2863
Telex 011 7595
Cable Biotron Baires

BRAZIL

Hewlett Packard do Brasil
I e C Ltda
Alameda Rio Negro 750
[illegible]
06400 **Barueri** SP
Tel 429 3222
Cable HEWPACK Sao Paulo

Hewlett Packard do Brasil
I e C Ltda
Rua Padre Chagas 32
90000-**Pôrto Alegre**-RS
Tel (0512) 22-2998 22-5621
Cable HEWPACK Porto Alegre

Hewlett-Packard do Brasil
I e C Ltda
Av Epitacio Pessoa 4664
Lagoa
20000 **Rio de Janeiro**-RJ
Tel
Telex 021-21905 HPBR-BR
Cable HEWPACK
Rio de Janeiro

Olympia (Chile) Ltda
Rodrigo de Araya 1045
Casilla 256 V
Santiago
[illegible]
[illegible]
[illegible]
[illegible]

Instrumentacion
H A Langebaek & Kier S A
Carrera 63 No 49-A 31
Apartado 54098
Medellin
Tel 304475

COSTA RICA

Cientifica Costarricense S A
Avenida 2 Calle 5
San Pedro de Montes de Oca
Apartado 10159
San Jose
Tel 24 38-20 24-08 19
Telex 2367 GALGUR CR
Cable GALGUR

ECUADOR

Computadoras y Equipos
Electrónicos del Ecuador
P O Box 6423 CCI
Eloy Alfaro No 1824 3 Piso
Quito
Tel 453 482
Telex 2548 CYEDE ED
Cable CYEDE Quito

Medical Only
Hospitalar S A
Casilla 3590
Robles 625
Quito
Tel 545-250
Cable HOSPITALAR Quito

EL SALVADOR

IPESA
Bulevar de los Heroes 11-48
San Salvador
Tel 252787

GUATEMALA

IPESA
[illegible]
[illegible]
Guatemala City
Tel 316627 314786 664715 ext 9
Telex 4192 Teletro Gu

MEXICO

Hewlett-Packard Mexicana
S A de C V
Av Periférico Sur No 6501
Tepepan Xochimilco
Mexico 23 D F
Tel 905-676 4600
Telex 017-74 507

Hewlett Packard Mexicana
S A de C V
Ave Constitucion No 2184
Monterrey N L
Tel 48-71-32 48 71 84
Telex 038-410

PANAMA

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
Ciudad de Panama
Tel 64 2700
Telex 3483103 Curundu
Canal Zone
Cable ELECTRON Panama

PERU

Compañia Electro Médica S A
Los Flamencos 145
[illegible]
Lima
Tel 41-4325
Telex Pub Booth 25424 SISIDRO
Cable ELMED Lima

TRINIDAD & TOBAGO

CARTEL
Caribbean Telecoms Ltd
P O Box 732
69 Frederick Street
Port-of-Spain
Tel 62 5[illegible]

VENEZUELA

Hewlett Packard de Venezuela
C A
P O Box 50933
Caracas 105
Los Ruices Norte
3a Transversal
Edificio Segre
Caracas 107
Tel 239 4133 (20 lines)
Telex 25146 HEWPACK
Cable HEWPACK Caracas

FOR AREAS NOT LISTED CONTACT

Hewlett Packard
Inter-Americas
3495 Deer Creek Rd
Palo Alto California 94304
Tel (415) 856 1501
TWX 910 373 1260
Cable HEWPACK Palo Alto
Telex 034 8300 034 8493

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 CORVALLIS, OREGON

Postage will be paid by

Hewlett-Packard
1000 N.E. Circle Blvd.
Corvallis, Oregon 97330
Attn. Technical Publications Manager

HEWLETT
PACKARD

00085-90002 Rev. C 7/80

Printed in U.S.A.

HEWLETT-PACKARD

I/O ROM

OWNER'S MANUAL

SERIES 80

CHEMICALS
82936A ROM DRAWER
HEWLETT PACKARD
MADE IN U.S.A.
Z =
X = (A^T A)^-1 A^T
A =
K =
R =

HEWLETT
PACKARD

I/O ROM
Owner's Manual

Series 80

January 1983

00087-90121

Printed in U.S.A.

Hewlett-Packard Company 1983

Contents

Section 1: An Introduction to I/O 3
- Introduction 3
- Installing the I/O ROM 3
- Removing the I/O ROM 4
- The Job of an Interface 5
- Printing to Peripheral Devices 8

Section 2: Simple I/O Operations 9
- Introduction 9
- Using Simple OUTPUT Statements 9
- Using Simple ENTER Statements 10

Section 3: Formatted I/O Operations 13
- Introduction 13
- Formatted OUTPUT 13
- Formatted ENTER 18
- Advanced Use of Terminator Images 21
- A Word of Advice About Images 24
- Converting I/O Data 25

Section 4: Error Handling 29

Section 5: Specialized Transfers 31
- Introduction 31
- Using Buffers 36
- Data Transfers 40

Section 6: End-Of-Line Branching 45
- Some Background on Interrupts 45
- End-of-Line Branch Programming 45

Section 7: Keyboard Control 55
- Introduction 55
- Key Mask Programming 55

Section 8: Direct Interface Communication 59
- Introduction 59
- Checking the Status 59
- Interface Control 61

Section 9: Additional I/O Commands 63
- Interface-Dependent Statements 63
  - The SEND Statement 63
  - HALT, ABORTIO, and RESET 63
  - Bus-Controlling Functions 64

Section 10: Binary Functions 65
- Introduction 65
- Review of Base 2 65
- Review of Logical Operations 66
- The Binary AND Function 69
- The Binary Inclusive OR Function 69
- The Binary Exclusive OR Function 69
- The Binary Complement Function 70
- The Bit Test Function 70

Section 11: Base Conversion Functions 73
- Introduction 73
- Review of Alternate Representations 73
- Conversions From Base 10 to an Alternate Base 74
- Conversions From an Alternate Base to Base 10 75
- Converting From One Alternate Base to Another 76

Appendix A: Syntax Reference 77

Appendix B: Maintenance, Service, and Warranty 133

Appendix C: Error Messages 137

Appendix D: ASCII Character Set 141

Section 1

An Introduction to I/O

Introduction

The power of your HP Series 80 Personal Computer is greatly extended by its ability to monitor and control external events. I/O, which stands for Input/Output, refers to this process of interaction between the external, or *peripheral,* device and the computer. There are several interfaces available so that practically any device designed for computer control may be attached and thus monitored and controlled.

When the computer is monitoring an event it is said to be the *listener* and it receives a message from the talker or *source*. This data message is referred to as *Input*. When the computer is controlling, on the other hand, it is the *source* and the peripheral device that it is talking to is called the *destination*. In this case, the message is considered *Output*.

The process of transferring data can become rather involved, depending on the type of peripheral and the application. Fortunately, a thorough understanding of interfacing techniques is not necessary to be able to utilize the I/O ROM with your computer; hence, for maximum effectiveness, this manual is arranged into three parts:

1. getting started with I/O—how to make use of simple I/O features quickly and easily (sections 1-3);
2. undertaking more complex I/O operations—understanding the full capability of I/O ROM interface commands (sections 4-10);
3. appendices—references and useful information in condensed form.

Sections 1 through 3 are adequate to explain operation of the printer interface. Also, a number of examples are presented in the HP 82949A Printer Interface Owner's Manual. If you intend to use one of the additional interfaces it is recommended that you become familiar with the additional flexibility of the I/O ROM as presented in sections 4 through 10. The owner's manuals for the Serial, HP-IB, GPIO, BCD, and other interfaces will then provide detailed instructions for each particular interface.

This manual covers both the I/O ROM for the HP-85 and HP-83 (part number 00085-15003) and the I/O ROM for use with the HP-87 and HP-86 (part number 00087-15003).

Installing the I/O ROM

Use of the I/O ROM requires an HP 82936A ROM Drawer. The ROM Drawer is a plug-in module that contains six rectangular slots for individual plug-in ROMs. Any HP Series 80 ROM will fit in any of the six positions in the ROM Drawer.

The HP Series 80 Personal Computer has four ports located in the back. These ports are used to hold extra memory, option ROMs, and interfaces. The I/O ROM should be installed in a ROM Drawer, which is then installed in one of the computer ports.

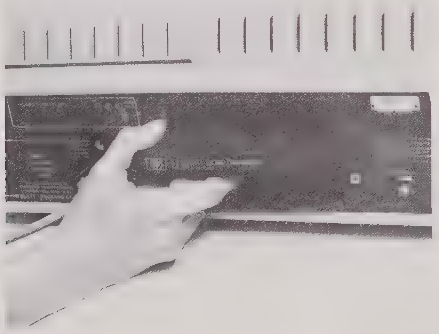
CAUTION

Do not remove or install the ROM Drawer or any plug-in module while the computer power is switched on. Failure to switch off the computer power may cause damage to the module or the computer.

The installation procedure is as follows:

1. If the ROM Drawer is already plugged into the computer, TURN OFF power to the computer and remove the ROM Drawer.
2. Remove the plastic cover from an empty ROM socket by inserting the eraser end of a pencil into the circular hole on the underside of the cover.
3. Align the ROM so that its chamfered end matches the chamfered end of the socket. Press the ROM lightly into the socket until it is even with the top of the ROM Drawer.

4. With the ROM labels facing up, press the ROM Drawer firmly into one of the ports in the back of the computer. The port and module are keyed so that the ROM Drawer cannot be installed upside down.



Removing the I/O ROM

The procedure to remove a ROM is as follows:

1. TURN OFF power to the computer.
2. Remove the proper ROM drawer.
3. Turn the drawer over and remove the ROM by using a pen, pencil, or small screwdriver to push gently through the hole underneath the ROM.

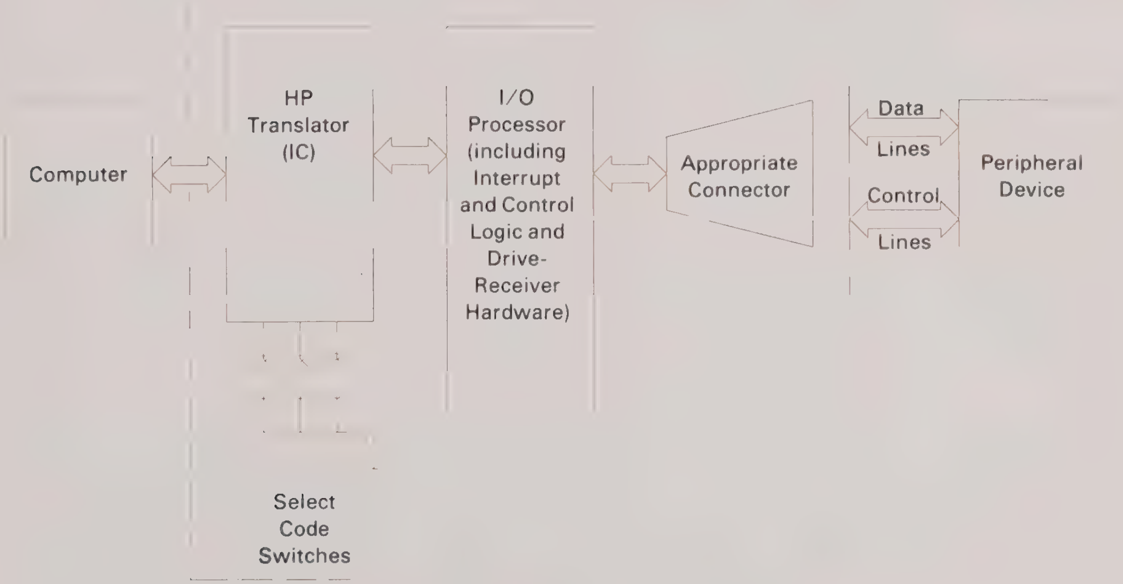
The Job of an Interface

An interface is the hardware link that is needed to allow efficient communication with peripheral devices. The job of an interface is to provide compatibility in four major areas. These are:

- Mechanical Compatibility
- Electrical Compatibility
- Data Compatibility
- Timing Compatibility

The following diagram shows the basic hardware components of an interface in its position between the computer and the peripheral device.

Interface Functional Diagram


Computer
HP Translator (IC)
I/O Processor (including Interrupt and Control Logic and Drive-Receiver Hardware)
Appropriate Connector
Data Lines
Control Lines
Peripheral Device
Select Code Switches

Mechanical and Electrical Compatibility

Mechanical compatibility simply means that the plugs and connectors must fit together. The HP 829XX Series plug-in modules are designed to be mechanically compatible with your computer. Certain interfaces, like HP-IB, are always mechanically compatible with their peripheral devices. Other interfaces, like the 16-bit parallel interface (GPIO), are supplied without peripheral connectors. In these cases, it becomes your responsibility to install a mechanically compatible connector. If you need to do this, study the owner's manual supplied with your interface. Electrical compatibility means that the interface must change the voltage and current levels used by the computer to those used by the peripheral device. The *Translator IC* used in each interface and the *Drivers* and *Receivers* insure that all HP interfaces are compatible with the computer, and usually compatible with the peripherals. If you have questions about electrical compatibility with your peripheral device, study the interface owner's manual and the peripheral device documentation.

Data Compatibility

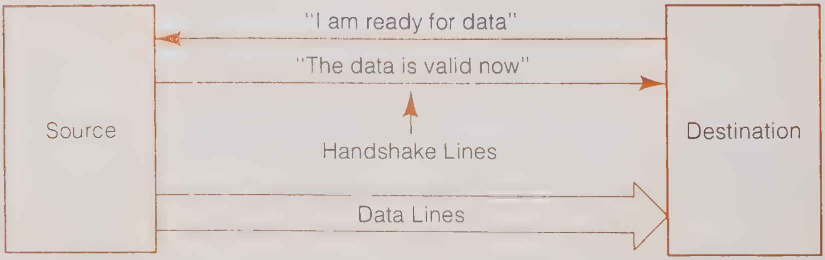
Mechanical and electrical compatibility alone do not guarantee that the computer and peripheral device will be able to communicate. Another requirement is that both devices must understand the data being sent by the other. Just as two humans who do not speak the same language need a translator, messages between the computer and the peripheral device may require some form of translation. The computer, with its versatile programming capability, usually performs this function. However, this job is sometimes given to the interface. The BCD interface is one example of giving the translation process to the interface. To handle those cases where the computer performs the data translation or conversion, the I/O ROM provides a wide variety of special functions and conversion capabilities. These capabilities are covered in subsequent sections of this manual.

Timing Compatibility

Computers and their peripheral devices have such a wide range of operating speeds that an orderly mechanism is required for successful transfer of data. This timing mechanism is referred to as *handshake*. Although there are different varieties of handshake, the basic sequence can be summarized as follows:

1. The receiver signals that it is ready for an item of data, then waits for a signal from the sender that the data is available.
2. The sender outputs an item of data and signals the receiver when the data is available.
3. When this "data available" signal is recognized, the receiver inputs the data and signals that it is busy with this input operation.
4. The sender waits until the receiver is ready before it makes a new item of data available. When the receiver is ready, this process repeats.

The following simplified diagram further illustrates the general concept of handshaking.



The *I/O Processor* is a component of each interface, which is itself a microcomputer. It is supplied with a small set of programs limited to data conversion and handshake management routines. Through the use of various commands available with the I/O ROM, the I/O Processor can be reconfigured to perform several different types of handshake operations.

Choosing the Source or Destination

To send someone a message through the mail, you must specify their address before the post office will even attempt delivery. So it is when you want to communicate with peripheral devices. The device with which you want to communicate must be specified within your program. This selection process is called *addressing*. The HP Series 80 Personal Computer addresses its peripheral devices through the use of a *device selector* in the I/O statements. Choosing the proper device selector depends upon the interface used and the way it is used. The following two discussions detail the two types of device selectors.

Using Interface Select Codes

If you are not using an HP-IB interface, and you have only one device connected to the interface, the device selector can be simply the *select code* of the interface. An interface select code is analogous to a house number in a mailing address. It is a number between 3 and 10 (inclusive) that identifies the interface. Each type of interface is set to a different select code at the factory. The following table summarizes these factory settings.

HP Part Number	Name	Select Code Setting
82937A	HP-IB	7
82938A	HP-IL	9
82939A	Serial	10
82940A	GPIO (Parallel)	4
82941A	BCD	3
82949A	Printer	8
82950A	Modem	10

Note: For electrical reasons, you must never plug in two interfaces that are set to the same select code.

Serious electrical conflicts can result if the interfaces and select codes do not correspond uniquely. In other words, if there are two interfaces present with the same select code, neither one of them will work. As you can see from the preceding table, the factory settings prevent this from happening unless you are using two interfaces of the same type. If you need to use two interfaces that came with the same factory setting, you must change the select code on one of them. The procedure for changing a select code is covered in the owner's manual for your interface. Follow its instructions carefully.

Using a Primary Address

If you are using an HP-IB interface, or if you have more than one device connected to an interface, the device selector is a 3-digit or 4-digit number formed from the interface select code and a *primary address*. This method of addressing is like mailing a letter to someone in an apartment building. Giving the street address will get the letter to the right building, but you still need to specify an apartment number to get the letter to the final destination. When a primary address is used, it is analogous to the apartment number. It identifies a specific device to be selected from a group of devices serviced by one interface. Some examples:

- A device selector of **721** specifies device **21** on interface **7**.
- A device selector of **301** specifies device **1** on interface **3**.
- A device selector of **1002** specifies device **2** on interface **10**.

Printing to Peripheral Devices

One of the simplest ways to direct the computer's output to a peripheral device is the `PRINTER IS` statement. The I/O ROM provides the capability of printing to external devices by using statements such as `PRINTER IS 4` or `PRINTER IS 720`. Any valid device selector can be used with the `PRINTER IS` statement. The same holds true for the `CRT IS` statement.

The `PRINTER IS` device is the destination for the output from all `PRINT` and `PLIST` statements. The `CRT IS` device is the destination for the output from all `DISP`, `LIST`, and `CAT` statements, as well as all "Error" and "Warning" messages. When programs are listed to a peripheral device, each program line is output as a single string. For the HP-85 this means that there is no indenting or 32-character wraparound as occurs when listings are done on the internal printer or CRT. `PRINT` and `DISP` both print out with the default line length set by the computer. To extend the line length or utilize more elaborate formatting you will need to explore the next two chapters. If, on the other hand, your only I/O requirement is to direct program listings or the output from `PRINT` statements to an external device, you need not read any further. Simply connect the desired interface and include the appropriate `PRINTER IS` statement in your program.

Section 2

Simple I/O Operations

Introduction

Section 1 talked about performing output operations with `PRINTER IS` and `PRINT` statements. Although this simple technique is very handy, it falls short of the mark in many circumstances. The most obvious shortcoming is that there is no corresponding `KEYBOARD IS` statement to allow input from external devices. Even when output is the only desired operation, it can be very inconvenient re-specifying the `PRINTER IS` device all the time when a program communicates with multiple peripheral devices.

The principal tools for using interfaces to move data in and out of the computer are the `OUTPUT` and `ENTER` statements. These statements are the core of I/O operations. They are usually the fastest and easiest ways of getting data from the source to the destination in its final form. Many applications require no more than the proper use of `OUTPUT` and `ENTER`.

Simple `OUTPUT` and `ENTER` statements (as described in this section) use ASCII representation for all data. *ASCII* stands for *American Standard Code for Information Interchange*. It is a commonly used code for representing letters, numerals, punctuation, and special characters. The ASCII code provides a standard correspondence between binary codes that are easily understood by the computer and alphanumeric symbols that are easily understood by humans. A complete list of the characters in the ASCII set and their code values is included in appendix D.

When special formatting is desired or when binary code is handled directly without using ASCII representation, the `OUTPUT USING` and `ENTER USING` forms are very convenient. These forms are discussed in section 3.

Using Simple OUTPUT Statements

A simple `OUTPUT` statement can be used anywhere that a simple `PRINT` statement is proper. The `OUTPUT` statement contains the device selector(s) of the destination device(s) and a list of the items to be output. The primary difference between `OUTPUT` and `PRINT` is that `PRINT` statements do not contain a device selector. Here are some examples of properly written `OUTPUT` statements:

```
OUTPUT 1 ; "Hello"
OUTPUT 3 ; X
OUTPUT S1 ; A$,B$
OUTPUT 703,725 ; X;Y;Z
OUTPUT 1000 ; A(1);B(3),N$[2,7]
```

Notice that a semicolon is used to separate the device selector from the output list, and commas or semicolons can be used to separate items within the output list. Items in the output list may be numeric variables, numeric constants, string variables, or string constants. A Carriage Return/Line Feed (End of Line sequence) is output after the last item in the output list.

The difference between using a comma and a semicolon to separate items in the output list is the spacing, or *field* of the items. The simple `OUTPUT` statement uses the same field as the `PRINT` statement. The semicolon calls for a compact field, while the comma produces free field. These fields are summarized in the following table.

	Numeric Data	**String Data**
Compact Field ; (*semicolon*)	Digits of the number are output, preceded by a space (if plus) or a minus sign (if minus), and followed by one space.	Characters of the string are output with no leading or trailing spaces.
Free Field , (*comma*)	Digits of the number (with leading space or minus sign) are output left-justified in a field of 11, 21, or 32 characters. Trailing spaces are output as necessary to fill the unused portion of the field.	Characters of the string are output with no leading spaces and no more than 20 trailing spaces.

The actual field width is determined by the same process used when items are output with the PRINT statement. Therefore on the HP-85, the computer pretends that it is displaying items on the CRT and sets a field width that would cause items to start in column 1 or 22 of the 32-column display. On the HP-87 the items will start in columns 1, 22, 43, or 64 of the 80-column field. If this is an undesirable format, you may need to separate items in the output list with semicolons or use formatted output as explained in the following section.

Using Simple ENTER Statements

A simple `ENTER` statement can be used anywhere that an `INPUT` statement is proper. The `ENTER` statement contains the device selector of the source device and a list of items to be entered. Remember that `INPUT` statements always use the keyboard as the source and contain no device selector, while `ENTER` statements always use a peripheral device as the source and contain the device selector of that device. Here are some examples of properly formed `ENTER` statements:

```
ENTER 3 ; X
ENTER S1 ; A$,B$,C$
ENER 703 ; X,Y,Z
ENTER 1000 ; A(1),B(3),N$
```

Notice that a semicolon is used to separate the device selector from the enter list and commas are used to separate items within the enter list. Items in the enter list may be numeric variables or string variables.

To use the `ENTER` statement effectively, it is important to understand what constitutes the beginning and ending of an entry into a variable. The simple `ENTER` statements just shown use a "free field format" for processing incoming characters. This format operates differently with string and numeric data.

Entering Numeric Data

The computer enters numeric values by reading the ASCII representations of those values. For example, if the computer reads an ASCII "1", then an ASCII "2", and finally an ASCII "5", it places the value one hundred twenty five into a numeric variable.

Understanding the process that the computer uses to read a free field number can help you remove much of the mystery from I/O. Suppose your program has the statement:

```
ENTER 3 ; X,Y
```

Now assume that when this statement is executed, the following character sequence is received through the interface at select code 3:

T	U	E	S	D	A	Y		D	E	C		1	1	,		1	9	7	9	cr	lf

The computer ignores all leading non-numeric characters, so the "TUESDAY DEC" characters do nothing. Then the "11" is read. Once the computer has started to read a number, a non-numeric character signals the end of that number. Therefore, the comma after the 11 causes the computer to place the value eleven into X and start looking for the next value. The space in front of "1979" is ignored and the computer reads the "1979". The carriage-return character causes the computer to place the value nineteen hundred seventy nine into variable `Y`. Finally, the computer keeps reading until it finds a line feed character. This terminates the `ENTER` statement, so the computer goes on to the next program line with `X=11` and `Y=1979`.

The process just described can be easily summarized. When entering numeric data using free field format, the computer:

1. Ignores leading non-numeric characters.
2. Ignores all spaces—leading, trailing, or imbedded.
3. Uses numeric characters to build a numeric value.
4. Terminates the building of a value when a trailing non-numeric character is encountered.
5. Inputs characters until a line feed character is encountered.

The discussion so far has referred to numeric and non-numeric characters without being specific. The digits 0 thru 9 are always numeric characters. Also, the decimal point, plus sign, minus sign, and the letter E can be numeric if they occur at a meaningful place in a number. For example, assume that the following character sequence is read by an `ENTER` statement:

–	–	T	E	S	T		1	2	.	5	E	–	3

If a numeric value is being entered, the leading minus signs and the E in "TEST" will be ignored. They have no meaningful numeric value when surrounded by non-numeric characters. However, the characters "12.5E – 3" will be interpreted as 12.5×10^{-3}. In this case, the minus sign and the exponent indicator (E) occur in a meaningful numeric order, so they are accepted as numeric characters.

Entering String Data

The computer enters string data by placing ASCII characters into a string variable. The process used for free field entry is straightforward. All characters received are placed into the string until:

1. The string is full or,
2. A line feed character is received or,
3. A carriage return/line feed sequence is received.

Assume that the computer is executing the statement:

```
ENTER 4 ; A$,B$,C$
```

The following character sequence is received:

H	E	L	L	O	lf	cr	lf	T	H	E	R	E	cr	lf

The letters "HELLO" are placed into `A$` when the first line-feed is encountered. Note that the line-feed itself is not placed into `A$`; it acts only as a terminator for the entry into `A$`. Then the entry into `B$` begins. However, a carriage return/line feed sequence is read immediately. This terminates the entry into `B$`. Since neither the carriage return nor the line feed is placed into `B$`, `B$` becomes the null string. Next, the entry into `C$` begins. The characters "THERE" are placed into `C$`, terminated by the carriage return/line feed following those characters. With the enter list now satisfied and a line feed detected at the end of the data, the computer will go on to the next program line.

Note that carriage return characters are only ignored when they are immediately followed by a line feed character. If a carriage return is received and not followed by a line feed, the carriage return is placed into the string.

Another example can be used to show termination on a full string. This time, suppose the program contains the following statements:

```
DIM X$[3]
ENTER 4 ; X$
```

The following characters are sent to the computer:

B	O	Y	C	O	T	T	cr	lf

The computer places the characters "BOY" into `X$`, which fills the dimensioned length of 3. Then the computer continues to read the incoming characters until a line feed is encountered. At that time, the `ENTER` statement is completed, and the computer goes on to the next program step with `X$="BOY"`.

Section 3

Formatted I/O Operations

Introduction

Although free-field format works well for some I/O situations, there are times when more control over format is necessary. Perhaps the data is some binary pattern which has nothing to do with ASCII, or a line-feed terminator is not wanted or expected, or a column of numbers with the decimal points in line is desired, or numbers with only two exponent digits instead of three are required. There is a wide variety of reasons for desiring format control during I/O operations.

The format of information sent or received through interfaces is controlled by the use of *image specifiers*. These image specifiers can be placed in an image statement or can be included directly in an `OUTPUT` or `ENTER` statement. This section of the manual provides details on the meaning and use of image specifiers.

Formatted OUTPUT

An output image can control all major characteristics of output data, including spacing, appearance of the field, form of data representation, and use of end-of-line sequences. The computer uses an output image when some form of the `OUTPUT USING` statement is encountered. There are three forms of this statement:

1. `10 IMAGE` *<output image>*
   `20 OUTPUT` *ds* `USING 10 ;` *<output list>*
2. `OUTPUT` *ds* `USING` *<output image>* `;` *<output list>*
3. `10 FMAT: IMAGE` *<output image>*
   `20 OUTPUT` *ds* `USING FMAT;` *<output list>*

The examples above show the general forms of the `OUTPUT USING` statement. Here are some specific examples:

```
10 TOT: IMAGE "Total =",ZZ.D
20 IMAGE 5A,2X,17A
:
60 OUTPUT 4 USING TOT ; C1,C2,C3
70 OUTPUT 701 USING 20 ; A$,B$
80 OUTPUT 9 USING "#,B" ; X
90 OUTPUT S3 USING "MDDD.DD" ; T(1),T(2)
100 OUTPUT 710,711 USING I$ ; N$,A
```

In the general forms, the *ds* stands for "device selector". Device selectors are explained in Section 1. The symbol *<output image>* represents a proper list of image specifiers. The image specifier list may be a literal enclosed in quotes or the name of a string variable which contains the specifier list. The specifiers within the list must be separated by commas. The list of items to be output is shown by *<output list>*. It does not matter whether you use commas or semicolons to separate items within the list. All spacing is controlled by the image specifiers, so a semicolon has the same effect as a comma.

Numeric Images

The image specifiers in this group are used to control the form of numbers which are output. Most of these image specifiers are the same as the PRINT image specifiers that may already be familiar to you. Since there are many numeric images, these specifiers are broken down into three categories in the following discussion. The categories are digit characters, sign character, and punctuation characters.

Digit Characters

These are the image specifiers which form the digits of the number. They allow you to determine the number of digits before and after the decimal point, display or suppress leading zeros, and control the inclusion of exponent information.

Image Specifier	Meaning
D	Causes one digit of a number to be output. If that digit is a leading zero, a space is output instead. If the number is negative and no sign image has been provided, the minus sign will occupy one digit place. If any sign is output, the sign will "float" to a position just left of the left-most digit.
Z	Same as "D", except leading zeros are output.
*	Same as "D", except leading zeros are replaced by asterisks.
E	Causes the number's exponent information to be output. This is a 5-character sequence including the letter "E", the exponent sign, and three exponent digits.
e	Same as "E", except only two exponent digits are output.
K	Causes the number to be output in compact format. No leading or trailing spaces are output.

Sign Character

These are the image specifiers used to control the output of sign information. Note that if no sign specifier is included in the image, negative numbers will use a digit position to output the minus sign.

Image Specifier	Meaning
S	Causes the output of a leading plus or minus sign to indicate the sign of the number.
M	Causes the output of a leading space for a positive number or a minus sign for a negative number.

Punctuation Characters

These are the image specifiers used to control the output of punctuation within a number, such as the inclusion of a decimal point.

Image Specifier	Meaning
.	Causes an American radix point to be output (a decimal point).
R	Causes a European radix point to be output (a comma).
C	Usually placed between groups of three digits. Causes a comma to be output to separate the groups of digits (American convention).
P	Same as "C", except a period is used to separate the groups of digits (European convention).

The following examples show some of the ways of combining these specifiers and the resulting output when numbers are sent to a typical printer. Additional examples may be found in the "Printer and Display Formatting" section of the Series 80 owner's manuals and programming guides.

Example Statement **Printed Output**

```
OUTPUT 701 USING "ZZZZ.DD" ; 30.336                           0030.34
OUTPUT 701 USING "4Z.2D" ; 30.336                             0030.34
OUTPUT 701 USING "4Z.2D" ; -30.336                            -030.34
OUTPUT 701 USING "3DC3DC3D" ; 1E6                           1,000,000
OUTPUT 701 USING "3DC3DC3D" ; 1.2345E4                         12,345
OUTPUT 701 USING "3DC3DC3D" ; 1.2E9      (Overflow Error)      ,   ,
OUTPUT 701 USING "SZ.DDD" ; .5                                 +0.500
OUTPUT 701 USING "MZ.DDD" ; .5                                  0.500
OUTPUT 701 USING "MD.DDD" ; .5                                   .500
OUTPUT 701 USING "Z.DDE" ; .00456                           4.56E-003
OUTPUT 701 USING "Z.DDe" ; .00456                            4.56E-03
OUTPUT 701 USING "Z.DDe" ; -.00456                           -.45E-02
OUTPUT 701 USING "MZ.DDe" ; -.00456                         -4.56E-03
```

Notice in these examples that the image "ZZZZ" and the image "4Z" mean the same thing. The same is true for the "D" and "*" specifiers. You can indicate the number of digits desired by simply placing that number in front of the specifier. The use of parentheses, as in "3(D)", means something different. The image "3D" means "output one numeric quantity in a three-digit field". The image "3(D)" means "output three numeric quantities, putting each one in a 1-digit field".

Be careful of overflow conditions when using these image specifiers. An overflow occurs when the number of digits required to accurately represent a number is greater than the number of digits allowed for in the image. If this happens, a warning is issued and something is output so that the program can continue. However, exactly what is output is difficult to predict and will probably bear little or no resemblance to the number that caused the overflow.

String Images

The image specifiers in this group deal with the output of string characters. They can also be used in combination with the numeric image specifiers for spacing and labeling purposes. All of these image specifiers are the same as PRINT image specifiers that may already be familiar to you.

Image Specifier	Meaning
A	Causes the output of one string character. If all the characters in the current string have been used already, a trailing blank is output.
"*literal*"	A *literal* is a string constant formed by placing text in quotes, or by using the CHR$ function, or a combination of the two. The character sequence specified is output when a literal image is encountered. When the literal is enclosed in quotes, the quote marks themselves are not output. Literal images are commonly used for labeling other output. Literal images cannot be placed directly into OUTPUT statements. An IMAGE statement must be used if literal images are desired.
X	Causes the output of one space.
K	Causes the string to be output in compact format. No leading or trailing spaces are output.

The following examples show some of the many ways of using these specifiers and the resulting output when the characters are sent to a typical printer. Additional examples can be found in the "Printer and Display Formatting" section of each of the HP Series 80 owner's manuals.

```
Example Statements                                              Printed Output

OUTPUT 701 USING "5A,A" ; "X","Y"                                     X    Y
OUTPUT 701 USING "K,3X,K" ; "UNCLE","SAM"                         UNCLE   SAM
OUTPUT 701 USING "K,3X,K" ; 98.6,99.9                             98.6    99.9

10 IMAGE "TOTAL = ",3D,X,K
20 T=125 @ A$="CARS"
30 OUTPUT 701 USING 10 ; T,A$                               TOTAL = 125 CARS
```

Notice that the "X" and "A" image specifiers allow a number before them in the same fashion as the "D", "Z", and "*" specifiers. the "K" specifier works equally well with string data or numeric data. String and numeric image specifiers can be combined in the same image statement. If literal (string constant) images are desired, they must be placed in an `IMAGE` statement.

Binary Images

These image specifiers are not available without the I/O ROM, so they may not already be familiar to you. These images are used to cause information to be output as one or two binary bytes, rather than as a character representation. Sections 10 and 11 of this manual explain the details of binary (base 2) representation. If you are unfamiliar with binary numbers, it is suggested that you read section 9 before trying to use the binary image specifiers.

The items to be output using these images must be numbers in the proper range. If a value to be output is not an integer, it will be rounded to the nearest integer before being sent as a binary value.

Image Specifier	Meaning
B	Outputs a value as a single 8-bit byte. The value must be in the range of 0 thru 255. If the value to be output is out of range, the value modulo (256) is output.
W	Outputs a value as two 8-bit bytes comprising a 16-bit word. The most significant byte of the word is output first, followed by the least significant byte. The value to be output must be in the range of −32,768 to +32,767. Negative numbers are output in 16-bit 2's complement form. If the value to be output is out of range and positive, 32,767 is output. If the value is out of range and negative, −32,768 is output.

```
Example Statement                           Interface Output, Bit Pattern

OUTPUT 7 USING "B" ; 127                                         01111111
OUTPUT 7 USING "B" ; 3                                           00000011
OUTPUT 7 USING "W" ; 3                                  00000000 00000011
OUTPUT 7 USING "W" ; -1                                 11111111 11111111
```

Note that specifying a binary image does not automatically suppress the end-of-line sequence after the last byte is output. Therefore, in the examples just given, the bit pattern shown is output followed by a carriage return/line feed.

BCD interface users should note that "B" and "W" specifiers do not work with the HP 82941A BCD Interface. See the *BCD Interface Owner's Manual* for specific formatting examples.

End-of-Line Sequence Images

These image specifiers control the output of end-of-line sequences. An end-of-line sequence is one or more characters that is normally output after the last item in an output list, and/or a signal on an interface wire concurrent with the last byte output. Exactly which sequence or signal is used depends upon the programming of the interface responsible for the output. See your interface owner's manual for more details. If your program does not change the end-of-line sequence in the interface, the default is a 2-character sequence; a carriage return followed by a line feed. The following images do not alter the end-of-line sequence. They simply control whether or not it is output.

Image Specifier	Meaning
/	Causes the output of an end-of-line sequence. Often used for skipping lines in a printout.
#	Suppresses the output of the final end-of-line sequence. This specifier is frequently used with binary image specifiers to prevent the destination device from interpreting the end-of-line characters as binary data.

The "/" may be placed anywhere in the image list and may have a number before it to indicate how many EOL (end-of-line) sequences are desired. The "#" must be the first item in an image list and can only be specified once. Note also that the "#" only suppresses the EOL sequence that would ordinarily occur after the last item in the output list. It does not suppress any imbedded EOL sequences caused by the "/" specifier.

A typical use of the "#" image is to output one byte, and only one byte. The following statement does this:

```
OUTPUT 6 USING "#,B" ; X
```

This statement outputs the binary representation of X with no carriage return, line feed, or any other potentially unwanted bit patterns.

A typical use of the "/" image is shown by the statement:

```
OUTPUT 701 USING "K,4/,K" ; A$,B$
```

If the destination is a printer, A$ is printed, followed by three blank lines, then B$ is printed. If A$="HI" and B$="JOE", the character sequence output looks like this:

H	I	cr	lf	cr	lf	cr	lf	cr	lf	J	O	E	cr	lf

The "#" image specifier is unique to the I/O ROM. Its power is shown in the following example where repetitions of the PRINT statement cause printout on the same line.

```
10 FOR I=1 TO 3
20 PRINT USING "#,6A" ; "DOOBY"
30 NEXT I
40 PRINT USING "2A" ; "DO"
50 END
DOOBY DOOBY DOOBY DO
```

Formatted ENTER

Using `ENTER` statements with image specifiers gives you a high degree of control in two areas:

1. Accurately describing to the computer what the incoming data looks like and what should be done with it.
2. Precisely specifying what condition(s) constitutes the end point of an entry to a variable and the end point of the `ENTER` statement itself.

This discussion deals with data formatting images first, then presents the terminator images. The HP Series 80 Personal Computer uses an `ENTER` image when some form of the `ENTER USING` statement is encountered. There are three forms of this statement:

1. `10 IMAGE` *<enter image>*
   `20 ENTER` *ds* `USING 10 ;` *<enter list>*
2. `ENTER` *ds* `USING` *<enter image>* `;` *<enter list>*
3. `10 LBL: IMAGE` *<enter image>**
   `20 ENTER` *ds* `USING LBL ;` *<enter list>*

The examples above show the general forms of the `ENTER USING` statement. Here are some specific examples:

```
10 IMAGE 2(A),K
20 FMT: IMAGE 5D,2X,3De
:
60 ENTER 4 USING 10 ; A$,B$,X
70 ENTER 711 USING FMT ; I,J
80 ENTER 9 USING "#,B" ; A(1),A(2)
90 ENTER S2 USING "%,8A,/,K" ; Q$,R$
100 ENTER 712 USING I$ ; N$,A
```

The general forms use the same type symbols which were used to represent the `OUTPUT` statement. These are *ds* for "device selector", *<enter image>* for the list of image specifiers, and *<enter list>* for the list of variables to be entered. As with simple `ENTER` statements, the enter list must contain either string or numeric variables.

Data Images

The image specifiers in this group are used to tell the computer what to do with the incoming data stream. The basic choices are:

1. Use characters to build a numeric variable.
2. Place characters into a string variable.
3. Input bytes as binary values.
4. Skip over a number of characters.

Numeric Image Specifiers

These specifiers are used to control the input of numeric characters, including digits, sign, exponent, and punctuation.

---

*Note that labels are not allowed on the HP-85 or HP-83.

Image Specifier	Meaning
D Z * . S M	These specifiers all do the same thing. They tell the computer to accept one character to be used in building a numeric quantity. The incoming characters do not have to follow the specified format, there just has to be the right number of characters. The six different specifiers are provided so that your program can document the expected format of the characters, and so that `ENTER` and `OUTPUT` statements can share the same `IMAGE` statement, if desired.
E	Tells the computer to accept five characters to be used for building a number. The five characters do not have to be exponent information, but they can be.
e	Same as "E", except the computer accepts four characters to be used in building a number.
C	This specifier also tells the computer to accept one character to be used in building a numeric quantity. However, if a "C" is present **anywhere** in a number's image, all commas will be ignored while the number is being entered. Without this specifier, a comma would end the entry of a numeric quantity.
K	Tells the computer to enter a string or numeric variable using free field format (explained in Section 2).
R P	These specifiers are used with the `OUTPUT` statement to provide a European radix point and digit separator. However, these images are NOT permitted for an `ENTER` statement. If you need to enter numbers in European format, you can use the `CONVERT` statement (covered later in this section) to change the number into American format.

String Image Specifiers

These specifiers are used to enter characters into string variables.

Image Specifier	Meaning
A	Tells the computer to enter one string character.
K	Tells the computer to enter a string or numeric variable using free-field format (explained in section 2).

Some examples are in order. Suppose the following character sequence is received by the computer:

1	2	3	4	H	E	L	L	O	cr	lf

Any of the following `ENTER` statements can be used to enter a numeric variable followed by a string variable:

```
ENTER 720 USING "4D,5A" ; X,Y$
ENTER 720 USING "Z.DD,5A" ; X,Y$
ENTER 720 USING "e,K" ; X,Y$
```

Notice that any numeric image that accepts four characters will properly enter the "1234". String data can be entered with an "**n**A" image if **n** (number of characters) is known, or with a "K" if the number of characters is unknown.

Suppose instead that the incoming data was:

1	,	2	3	4	H	E	L	L	O	cr	lf

The `ENTER` image would now have to include a `"C"` for the entire "1234" to be entered. For example:

```
ENTER 720 USING "C4D,K" ; X,Y$
ENTER 720 USING "DDDDC,5A" ; X,Y$
```

Notice that the `"C"` does not have to appear at the same place in the image as the comma does in the incoming data. However, the comma is counted as a character.

Binary Image Specifiers

These specifiers are used to enter data that is received in binary format.

Image Specifier	Meaning
`B`	Tells the computer to enter one byte of binary data and enter its equivalent decimal value into a numeric variable.
`W`	Tells the computer to enter two bytes of binary data to be used in building a 16-bit, 2's complement binary word. The equivalent decimal value of the resulting word is entered into a numeric variable. The first byte entered is used as the most significant byte of the word.

Skipping Unwanted Characters

These specifiers can be used with incoming numeric or string data to skip over any characters not wanted for the input.

Image Specifier	Meaning
`X`	Tells the computer to skip over one character.
`/`	Tells the computer to skip to a line feed. Thus, after the variable has been satisfied, the computer "throws away" incoming characters until a line feed is received.

The `"X"` specifier should only be used when you have a good understanding of the structure of the incoming data, but can be very useful in formatting operations. For example, suppose that text is being entered from a remote computer that sends a line number at the beginning of every string. You know that the line number information always appears in the first eight characters of each string, and you don't want these line numbers in your data. The following format could be used to strip off the line numbers:

```
ENTER 720 USING "8X,K" ; A$
```

The `"/"` specifier is used to demand a line-feed field terminator before going on to the next variable. To see the effect of this specifier, assume that the incoming data is as follows:

1	2	3	H	I	lf	B	Y	E	cr	lf

Using the statement:

```
ENTER 720 USING "3D,K" ; X,Y$
```

causes `X` to get the value 123 and `Y$` becomes "HI". However, if the statement:

```
ENTER 720 USING "3D,/,K" ; X,Y$
```

is used, then `X` gets the value 123 and `Y$` becomes "BYE". The "`/`" specifier caused the computer to skip all characters after `X` was entered until it saw the line feed. Then the entry into `Y$` began with the first character after the line feed. Without the "`/`" specifier, the entry into `Y$` began as soon as the "`3D`" field was exhausted.

Eliminating the Line-Feed Requirement

The `ENTER` statement must "see" a line-feed character at the end of the incoming data before the program can go on to the next statement. If there is no line-feed character at the end of the data, the computer will be "hung up" waiting for one. If your incoming data does not have a line feed at the end, you can get the `ENTER` statement working properly by using an image specifier.

Image Specifier	Meaning
`#`	Eliminates the requirement for a line feed to terminate the `ENTER` statement. When this specifier is present, the `ENTER` statement terminates as soon as the last variable in the statement has been satisfied.

When the "`#`" specifier is used for this purpose, it must be listed as the first specifier in the image list. For example:

```
ENTER 3 USING "#,K" ; A$
ENTER 720 USING "#,4D,6D" ; X,Y
```

The first example statement shows an entry into a string variable using free-field format with the line-feed requirement removed. This statement terminates when the string is full. The second example shows a formatted entry into numeric variables with the line-feed requirement removed. This statement terminates after inputting ten characters.

Advanced Use of Terminator Images

The `ENTER` image specifiers discussed in the preceding sections are sufficient to handle the great majority of requirements. However, there are some special situations that demand an even greater amount of flexibility. Most of these special cases involve the EOI (End or Identify) line on the HP-IB. The following discussion is probably of no concern to most programmers. If you are one of those who must consider the EOI line, or if you have an unusual problem with line feeds, then read carefully. This is the most complex part of this section.

Field and Statement Terminators

The purpose of an `ENTER` statement is to read a *record*. To the programmer, a record is a logical grouping of data items. To the computer, a record is an incoming stream of data ended with a record terminator. Since the `ENTER` statement is ended when the record terminator is read, this manual refers to the record terminator as a statement terminator. If there is a requirement for a statement terminator in effect, the `ENTER` statement does not end until that terminator is received. (The action is slightly different when using buffers. These are covered in section 5.) If no terminator image is specified, the default statement terminator is a line feed character. To allow a carriage return/line feed sequence as a statement terminator, the I/O ROM ignores a carriage return if it is immediately followed by a line feed.

An incoming record often contains multiple *fields*. A field is the group of characters used to determine the input to a variable in the `ENTER` list. For example, an `ENTER` statement used to input a list of names and ages might look like this:

```
ENTER 720 ; N$,A
```

This statement reads a record containing a name and an age. This record has two fields. The first is a string field (the name), and the second is a numeric field (the age). A properly specified `ENTER` statement places the string field in `N$` and the numeric field in `A`.

It is not generally necessary to specify any terminator images to get the `ENTER` statement to perform properly. The system has built-in field terminators and a default statement terminator which are sufficient for most common applicants. These normal terminators are:

- A string field ends when the string is full (check your `DIM` statements), the character count from an image field is exhausted, or a line feed is received.
- A numeric field ends when any non-numeric character (except a space) is encountered or the character count from an image field is exhausted.
- The `ENTER` statement ends upon receipt of a line feed character or a carriage return/line feed sequence. This can be the same line feed that satisfied the last field in the `ENTER` list.

Given these normal terminating conditions, the `ENTER` statement mentioned previously properly separates the name field and the age field in two cases. One case is when there is a line feed separating the string field from the numeric field. The other case is when the string field is always of fixed length and `N$` is dimensioned to that length.

Terminator Images

If the normal terminating conditions are not ideally matched to your application, the use of terminator images can help solve the problem. The following image specifiers apply to both field and statement terminating conditions. Field terminators are the conditions that end the entry of data into a variable. Statement terminators are the conditions that end the `ENTER` statement after the last variable is satisfied.

Image Specifier	As a Field Terminator	As a Statement Terminator
`#`	Eliminates line feed as a terminating condition during free-field string entry. Line feeds entered are placed into the string.	Suppresses the requirement for a line-feed terminator. Statement ends when last field is satisfied.
`%`	Allows EOI as an additional terminating condition.	Allows EOI or line-feed as terminating conditions.
`#%` (or `%#`)	Allows EOI as an additional terminating condition, and also eliminates line feed as a terminator during free-field string entry.	Specifies that an EOI must be received to terminate the statement, and line feed is not a terminator.

Whether an image specifier controls statement terminator or field terminators depends upon where it is placed in the image. Consider the following example statement:

```
ENTER 720 USING "%,%K" ; X
```

When the terminator image is specified by itself as the first item in the image list (like the first `%`), it specifies the statement terminator. When the terminator image is combined with another specifier (like the `%K`), it specifies a field terminator. The `"#"`, `"%"`, and `"#%"` images all follow this convention.

Because the built-in field terminators are always in effect, these special terminator images only alter the system's action in a few cases. Let's look at each of these meaningful field terminating combinations individually.

Entering Line-Feeds Into a String

The image `"#K"` causes the computer to place all incoming characters (including line feeds) into a string until it is full. If there is a line feed forthcoming after the string is filled, this image is all that is necessary. If you wish the statement to end as soon as the string is filled (without waiting for a final line feed), the image `"#,#K"` should be used.

Using EOI to Terminate a String Entry

The image `"%K"` allows the computer to terminate a free-field string entry with the EOI signal. However, a device which uses EOI as its end-of-line indicator may not output any other end-of-line characters, like line feed. If this is the case, the proper image is `"%,%K"`. This allows the EOI signal to also terminate the ENTER statement. If you wish to enter line feed characters into the string and also wish to terminate with EOI, the image `"#%K"` can be used. This may need to be expanded to `"%,#%K"` if no line feed is expected to terminate the statement. The further expansion `"#%,#%K"` not only *allows* EOI to terminate the ENTER statement, but *requires* it as the only method of terminating the statement. Fixed-field entries can be checked for an expected EOI. For example, the image `"%7A"` inputs seven characters into a string and expects to have an EOI signal with the seventh character. Keep in mind that there are many valid combinations of these image specifiers. The combinations shown here are only some of the more common ones.

Using EOI to Terminate a Numeric Entry

The image `"%K"` allows the computer to terminate a free field numeric entry with the EOI signal. As mentioned in the preceding paragraph, the image `"%,%K"` may be necessary if the EOI signal is to terminate the ENTER statement also. Fixed field entries can be checked for an expected EOI. For example, the image `"%7D"` inputs seven characters to build a number and expects to have an EOI signal with the seventh character. Binary fields work in a similar manner. The image `"%W"` inputs two bytes to make a 16-bit integer and expects an EOI signal with the second byte.

There's Always an Exception

Not all terminator problems can be solved with terminator images. Consider again the example of a name field (string) followed by an age field (numeric). Suppose that the names are variable in length and separated from the age by a simple comma. If the ages came first, this would not be a problem since the comma would end the entry to the numeric variable. But since the string data is entered first in this example, the task is a bit trickier. You might be able to use a `CONVERT` statement (explained at the end of this section) to change the comma into a line feed and terminate the string that way. If the application does not permit the blanket conversion of commas to line feeds, then the entire record would have to be input into a temporary string variable. Once the record is entered, the `POS` function and string subscripts could be used to extract the name and age fields. This hypothetical situation emphasizes the importance of knowing the nature of the data you are trying to enter.

A Word of Advice About Images

Choosing the proper image for your application can often mean the difference between success and failure for your program. However, considering the wide range of peripheral devices and the near-infinite variety of possible data formats, it is understandably difficult to pick just the right image. Even experienced programmers will go through a period of trial and error before finding the perfect combination of image specifiers.

There is an old but true saying in the world of computers: "You can't program a computer to do something that you don't know how to do yourself." This is an appropriate statement for formatted I/O. If you don't know exactly what character sequence needs to be output or what an incoming sequence contains, it is very unlikely that you will know exactly what image specifiers to use.

Deciding on an exact character sequence for an output is simply a matter of definition. You know what data is generated by your program, so all you need to do is pick a desirable form for its output. The primary caution here is to avoid image overflow conditions.

But how can a programmer determine the exact nature of incoming data when he or she can't get it into the computer to study it? If the only tools available were the string and numeric image specifiers, this might be a significant problem. Fortunately, there is a way to inspect a totally unknown character sequence. Any sequence of bytes, including potential terminators, can be entered with the `"#,B"` image. The values that are printed or displayed are the decimal equivalents of the binary value for each byte. Admittedly, this is not the most convenient form of data to work with. However, you can use an ASCII table or the `CHR$` function to determine the exact character sequence which is being received. Then, knowing the exact nature of the incoming data, the job of choosing image specifiers will be much simpler. The following example program shows a typical use of this technique.

```
100 ! Program to inspect incoming data
110 S1=3 ! Interface select code
120 ! Estabish a terminating condition
130 SET TIMEOUT S1;3000
140 ON TIMEOUT S1 GOTO 230
150 !
```

```
160 I=1  ! Initialize counter
170 ! Input 1 byte; display analysis
180 ENTER S1 USING "#,B" ; X
190 DISP "BYTE";I;TAB(11);"VALUE =";X;TAB(24);"CHAR = ";
    CHR$(X)
200 I=I+1  ! Count bytes
210 GOTO 180
220 !
230 DISP "DATA INPUT HAS STOPPED"
240 RESET S1! Stop I/O operation
250 END
```

Converting I/O Data

The final type of formatting involves changing the data characters that are entered or output. An example cited earlier was incoming numbers in European format (with periods separating digit groups and a comma for the radix point). There is no image available to accept this type of data directly. The periods and commas need to be changed to other characters to give the computer what it wants. The tool for performing this kind of operation is the `CONVERT` statement. Its general form is:

`CONVERT` *<direction><select code><access method>* `;` *<string>*

The parameters are defined as follows:

<direction> Indicates whether the conversion is to take place during an `ENTER` (choose `IN`) or an `OUTPUT` (choose `OUT`).

<select code> Indicates which interface will use the conversion. Note that the `CONVERT` operation applies to all devices on a particular interface. A device selector is not allowed. The parameter must be an interface select code, range 3 thru 10.

<access method> This specifies the method of accessing the conversion table. The conversion table is a string variable, and there are two access methods. If `PAIRS` is specified, the string is treated as a list of character points. The second character of a pair is substituted for the first character whenever the incoming or outgoing character matches a first character. This method is a good choice when only a few characters need to be converted.

If `INDEX` is specified, the string is treated as a sequential look-up table. The numeric value of each incoming or outgoing character is used as an index into that table. The first element in the string corresponds to the character with a value of 0. If the value of the character to be converted is too large for the number of characters in the string, no conversion is performed. This method is a good choice when a large number of characters need to be converted.

<string> This represents the actual conversion table. It must be a string variable. A literal (string constant) is not allowed.

The use of the `CONVERT` statement should become more clear with a few examples. First, the European number format problem. This is a conversion for incoming data. One effective conversion is to replace a comma with a decimal point and replace a period with a space. The statements for doing this with an interface at select code 7 are:

```
A$=",.. "
CONVERT IN 7 PAIRS ; A$
```

The conversion has this effect:

12.345,6 is converted to 12 345.6

Since the free field format ignores spaces within a number and recognizes a decimal point, you do not even need an `ENTER` image to recognize the converted data. It is important to note that this `CONVERT` statement changes all periods to spaces and all commas to periods, whether they are part of a European number or simply part of a block of text. Since this could have some undesired effects, it is necessary to be able to turn off the conversion when it is no longer desired. The statement which cancels the conversion in this example is:

```
CONVERT IN 7
```

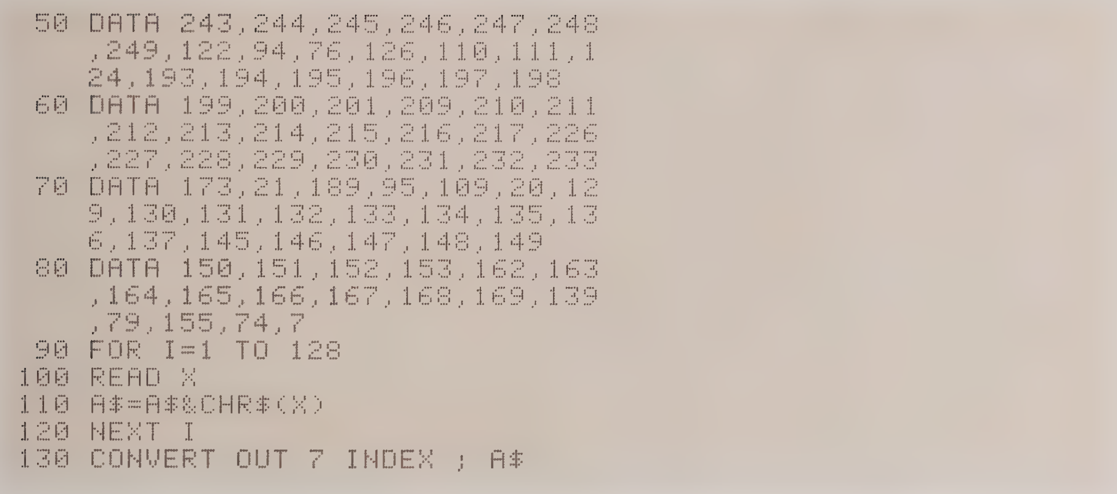
Giving only the direction and interface select code, without specifying `PAIRS` or `INDEX` or any other parameters cancels a previously selected conversion.

Control characters, such as carriage return or line feed, can also be converted. The following example shows the statements used to convert a carriage return to a line feed. This conversion is needed when entering data from a device which gives only a carriage return, without a line feed, as a delimiter.

```
A$=CHR$(13)&CHR$(10)
CONVERT IN 7 PAIRS ; A$
```

Another conversion example is the output of Extended Binary Coded Decimal Information Code (EBCDIC) instead of ASCII code. EBCDIC is another form of character representation used on certain types of computers. Since all the ASCII symbols have corresponding EBCDIC symbols, it is reasonable to choose the `INDEX` conversion mode using a string with 128 characters. In the following example, it is more important to understand the general process being used than to understand what the actual EBCDIC values are. The decimal equivalents of 128 EBCDIC characters are read from data statements and converted to string characters by the `CHR$` function. The resulting look-up table is included in a `CONVERT` statement for interface select code 7. The `INDEX` specifier tells the computer to use the outgoing ASCII character as an index to find the equivalent EBCDIC character. For example, an ASCII right brace (decimal value 125) will convert to a `CHR$(155)`, which is an EBCDIC right brace.

```
10 DIM A$[128]
20 A$=""
30 DATA 0,1,2,3,55,45,46,47,22,
   5,37,11,12,13,14,15,16,17,18
   ,19,53,61,50,38,24,25,63,36
40 DATA 28,29,30,31,64,90,127,1
   23,91,108,80,125,77,93,92,78
   ,107,96,75,97,240,241,242
```



```
 50 DATA 243,244,245,246,247,248
    ,249,122,94,76,126,110,111,1
    24,193,194,195,196,197,198
 60 DATA 199,200,201,209,210,211
    ,212,213,214,215,216,217,226
    ,227,228,229,230,231,232,233
 70 DATA 173,21,189,95,109,20,12
    9,130,131,132,133,134,135,13
    6,137,145,146,147,148,149
 80 DATA 150,151,152,153,162,163
    ,164,165,166,167,168,169,139
    ,79,155,74,7
 90 FOR I=1 TO 128
100 READ X
110 A$=A$&CHR$(X)
120 NEXT I
130 CONVERT OUT 7 INDEX ; A$
```

Section 4

Error Handling

Run-time errors on the HP Series 80 Personal Computer can be trapped by using the `ON ERROR` statement. You may already be familiar with the `ERRN` and `ERRL` functions which provide essential error information in a program. These functions can be used with the I/O ROM. However, all option ROMs share the error numbers starting at 101. So some other tools are necessary to identify the source of an error when more than one ROM is installed.

The I/O ROM provides two error functions in addition to the standard diagnostic capabilities of the HP Series 80 Personal Computer. These functions give the programmer the extra information necessary to isolate error conditions in a program.

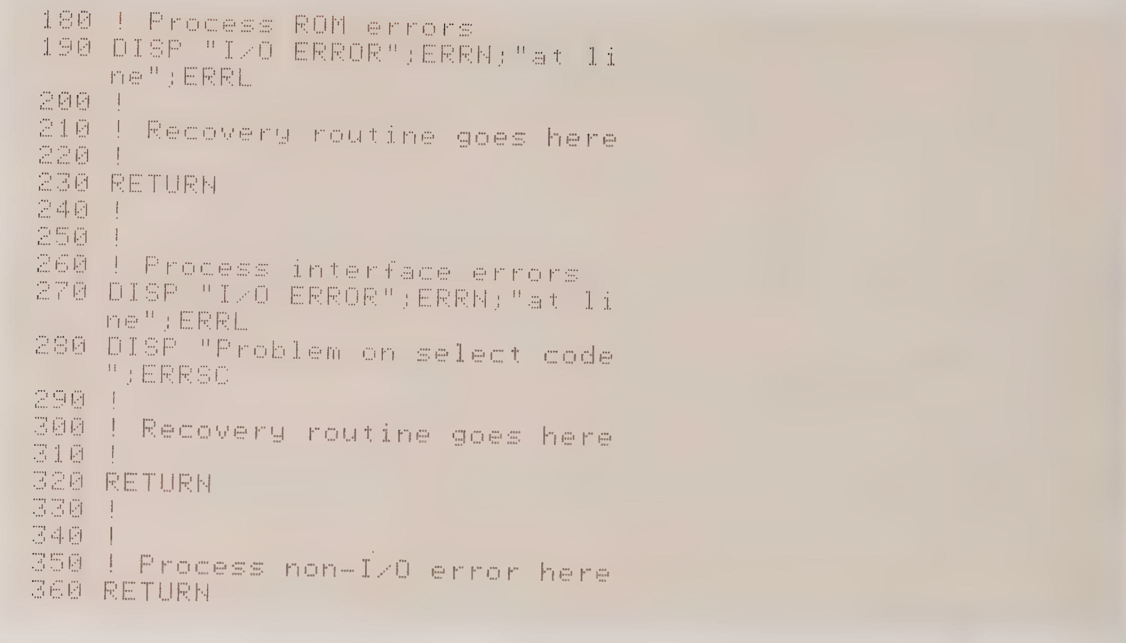
`ERROM` — Provides a number which identifies the option ROM which generated the most recent error. If the most recent error was caused by the I/O ROM, the `ERROM` function returns a value of 192. Note that this function is only updated by option ROM errors. Therefore, `ERROM` "remembers" the last option ROM error, even if the most recent error in the system was not caused by an option ROM.

`ERRSC` — Provides the select code of the interface which generated the most recent interface-dependent error. Note that this function is only updated by interface-dependent errors. Therefore, `ERRSC` "remembers" the last interface error, even if the most recent error in the system was not caused by an interface.

Because other option ROMs share similar error numbers to those of the I/O ROM, and because these functions are not updated by every system error, it is important to interrogate the various error functions in the proper order. If you are looking for I/O errors in an error recovery routine, check first for `ERRN>100`. If there is a ROM error, check `ERROM` to find which ROM. Having determined that the I/O ROM generated the error, check `ERRN` for an interface error before looking at `ERRSC`. Error numbers 101 and 112 will not occur during a running program. All other errors below 123 are interface-dependent errors. Therefore, a simple test for `ERRN<123` will tell if there was an interface error.

The following simple program segment shows the recommended order of function checks used to isolate I/O errors. This segment only displays an error message. An actual error recovery routine would also include statements to take whatever corrective action is appropriate in your specific situation.

```
 10 ON ERROR GOSUB 100
 20 !
 30 !
100 ! Test for non-I/O errors
110 IF ERRN<101 THEN GOTO 350
120 IF ERROM<>192 THEN GOTO 350
130 !
140 ! Test for interface errors
150 IF ERRN<123 THEN GOTO 260
160 !
170 !
```



```
180 ! Process ROM errors
190 DISP "I/O ERROR";ERRN;"at li
    ne";ERRL
200 !
210 ! Recovery routine goes here
220 !
230 RETURN
240 !
250 !
260 ! Process interface errors
270 DISP "I/O ERROR";ERRN;"at li
    ne";ERRL
280 DISP "Problem on select code
    ";ERRSC
290 !
300 ! Recovery routine goes here
310 !
320 RETURN
330 !
340 !
350 ! Process non-I/O error here
360 RETURN
```

There is a complete listing of all I/O errors, their meaning, and some debugging hints in appendix C.

Section 5

Specialized Transfers

Introduction

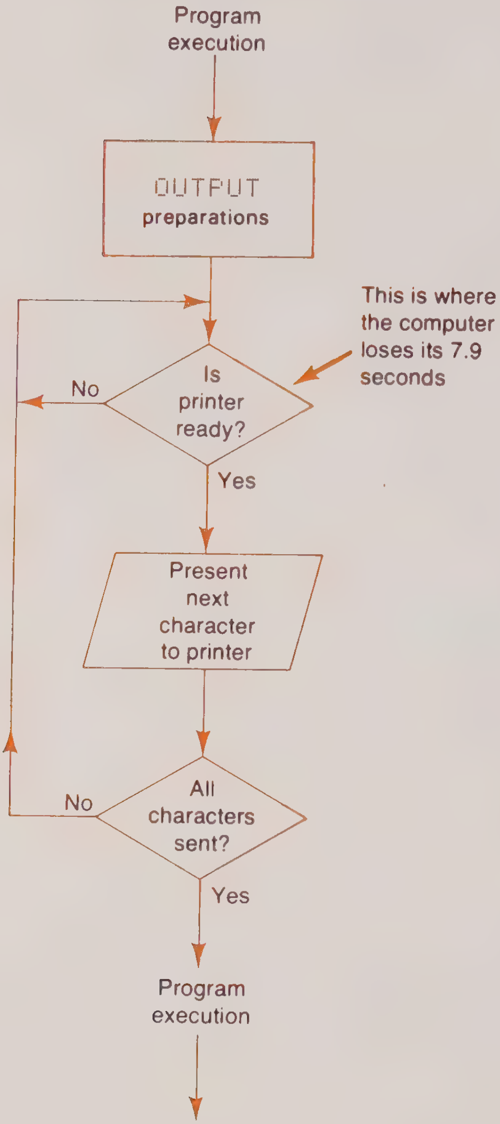
This section deals with data transfers as they are implemented by the TRANSFER statement. The basic purpose of the TRANSFER statement is to provide a flexible tool for moving data into and out of the computer. The key word here is flexibility.

This flexibility allows you to better match the computer's speed to that of the peripheral it is communicating with. Take the case of a very slow device, such as a 10 character-per-second printer. It takes such a printer 8 seconds to print an 80 character line, but our computer could send those same 80 characters in less than .1 second. If the computer is forced to wait on the printer, then the computer is losing 7.9 seconds of computation time out of every 8 seconds! The computer's power can obviously be increased by gaining back that 7.9 seconds. Let's see how.

The following diagrams constrast the default handshake method used by OUTPUT and ENTER with the *interrupt* method of TRANSFER. When the computer executes the OUTPUT statement (for example), it is forced to handshake *each* character of the data list until all the data has been sent. Only then is the computer free to execute the next program statement, about 8 seconds later. On the other hand, the interrupt TRANSFER statement sets up some special pointers to the data and enables the printer interface to interrupt the computer. Then the computer is free to execute the next program line, about 10 *milliseconds* later! (Enabling an interrupt is like hanging up a telephone receiver: the telephone is now able to "interrupt" you by ringing whenever someone calls.)

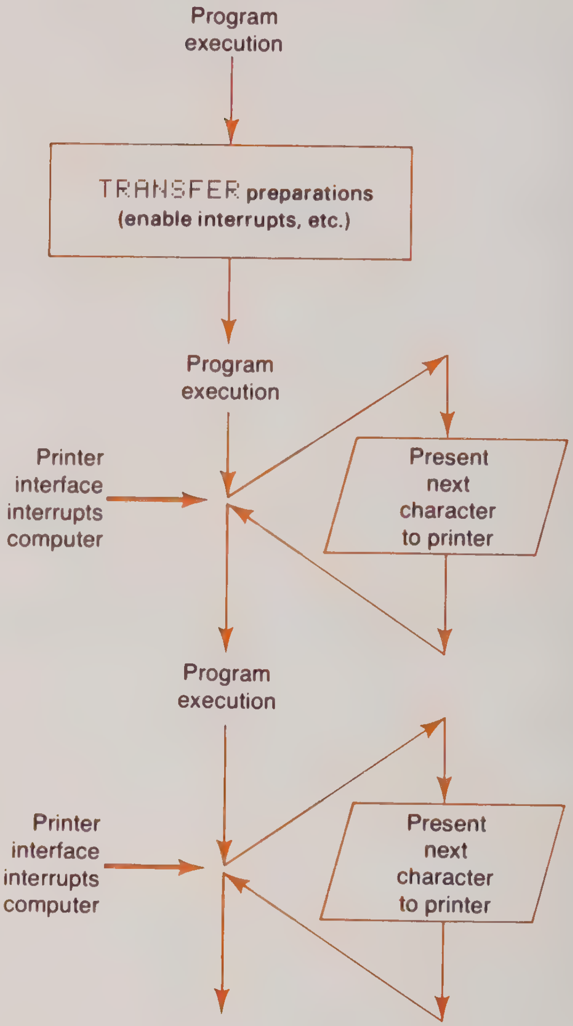
The computer continues program execution until the printer is ready for another character. The printer interface interrupts the computer from whatever it was doing, and the computer then fetches the next character, updates its pointers, check to see if all data has been sent, then continues on with what it was doing. If all the data has now been sent, the computer disables further interrupts from the interface (like taking the telephone receiver off the hook—no more rings) before continuing on with the program.

Handshake Method


Program
execution
OUTPUT
preparations
Is
printer
ready?
No
Yes
Present
next
character
to printer
All
characters
sent?
No
Yes
Program
execution

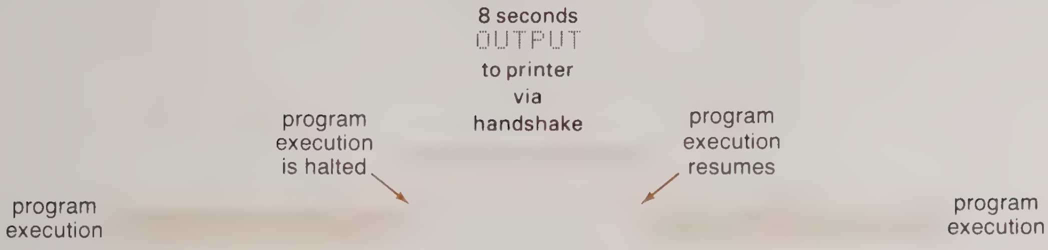
This is where the computer loses its 7.9 seconds

Interrupt Method

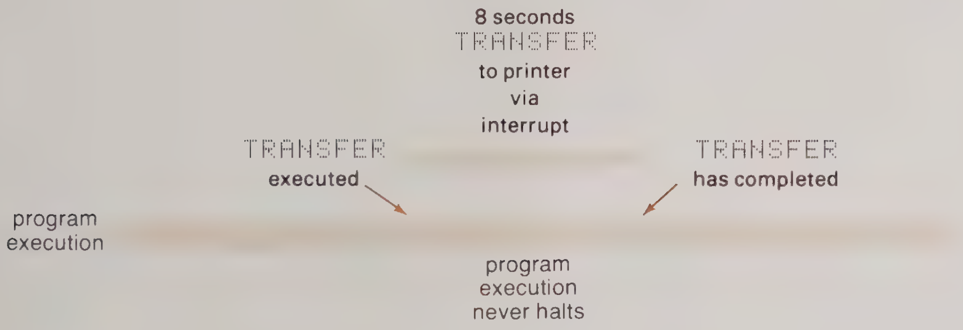

Program
execution
TRANSFER preparations
(enable interrupts, etc.)
Program
execution
Printer
interface
interrupts
computer
Present
next
character
to printer
Program
execution
Printer
interface
interrupts
computer
Present
next
character
to printer

If the two methods are looked at in a broader sense, it is possible to see the real difference: the handshake method is a linear, or sequential operation, while the interrupt method is a parallel, or overlap operation. Consider the following diagram:

Sequential


8 seconds
OUTPUT
to printer
via
handshake
program
execution
is halted
program
execution
resumes
program
execution
program
execution

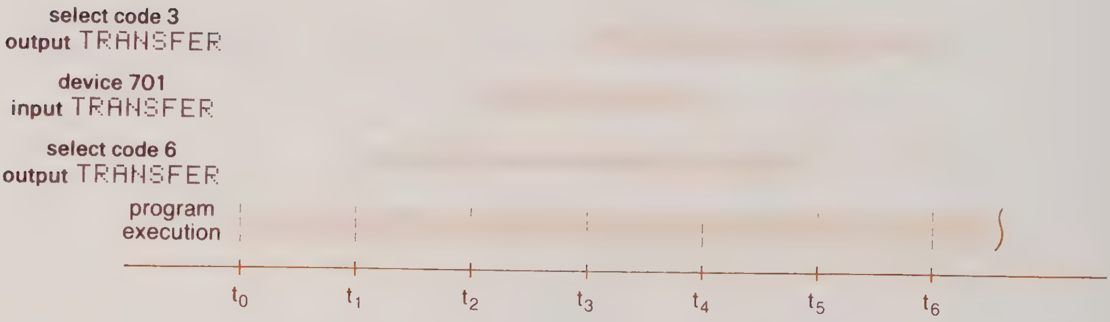
Overlap


8 seconds
TRANSFER
to printer
via
interrupt
TRANSFER
executed
TRANSFER
has completed
program
execution
program
execution
never halts

The sequential method effectively pauses the program for the duration of the OUTPUT operation, while the overlap method continues both with program execution and transfer operation.

An interesting possibility brought about by this overlap is that of multiple, simultaneous I/O operations. Suppose that the next program statement after the TRANSFER statement is *another* TRANSFER to a different device (a large-screen CRT monitor, for instance). Then *three* things are happening at once: the program is being executed, the printer is printing, and the external CRT monitor is displaying new characters.

The following diagram illustrates some of the power of overlapped I/O operations made possible with TRANSFER. (Please note that this is for interrupt transfer only, as explained later.)

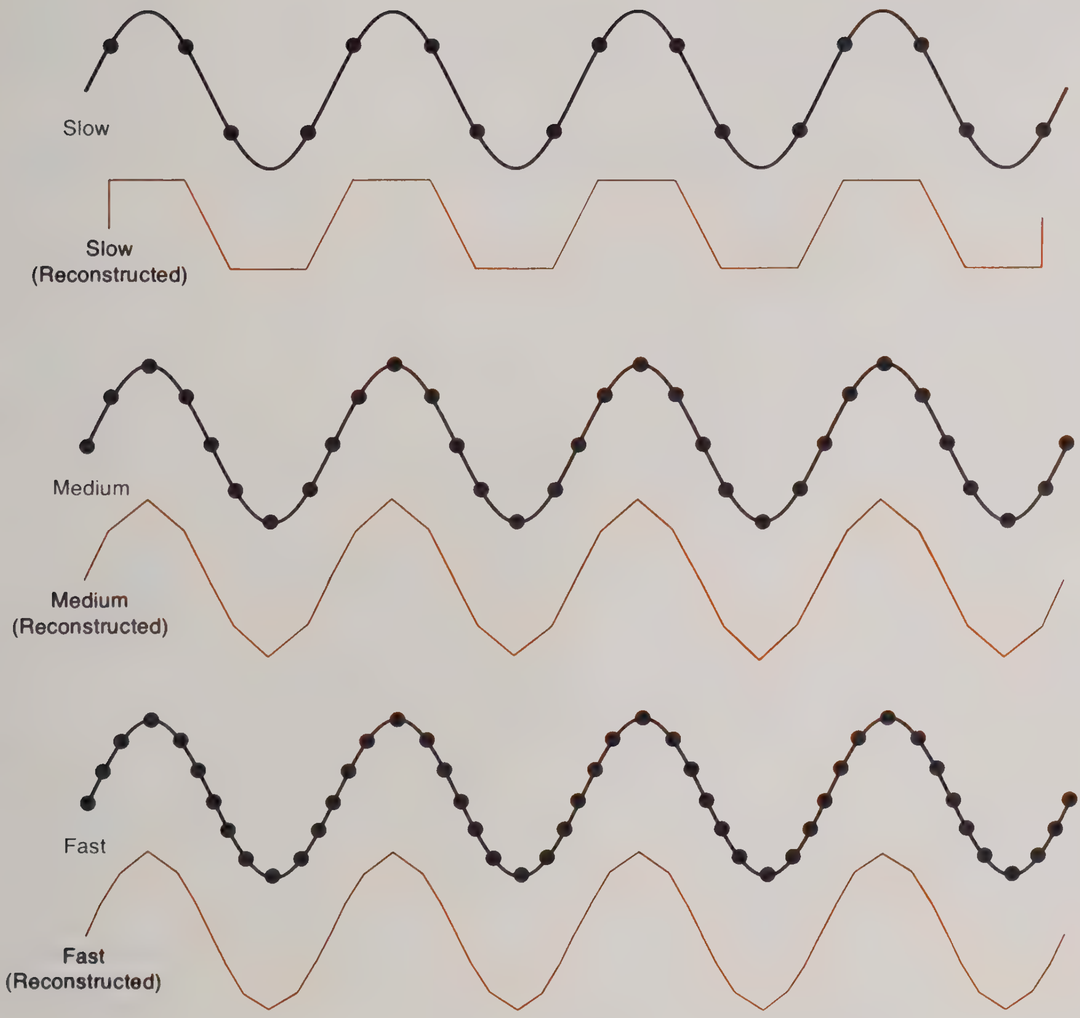


Program execution begins at time t_0, and some time later an output transfer to select code 6 is initiated (time t_1). At time t_2 an input transfer from HP-IB device 701 is started, and at time t_3 another output transfer is started, this time to select code 3. By now, three transfers and program execution are all going on at once. At time t_4, the input transfer terminates, and the two output transfers finish at t_5 and t_6. This overlap capability demonstrates some of the flexibility of system design made possible with the transfer statement.

There is another side to the transfer statements flexibility: speed. Certain operations are ineffective or impossible at slow or medium speeds, and require instead a high-speed transfer. Take the case where it is desired to analyze a signal's waveform by using an HP 3437A Voltmeter. This voltmeter is capable of producing a 3½ digit voltage reading up to 3600 times per second, which is a transfer rate of about 25,000 characters per second! The computer is not quite that fast, however; its transfer rate is closer to 20,000 characters per second.

The other type of TRANSFER statement is the *fast-handshake transfer*. This is an entirely different type of operation from the interrupt transfer. A fast handshake transfer represents the fastest possible data movement operation.

The following diagram illustrates the effect of sampling a signal at slow, medium, and high rates of speed (sample points are represented by dots):


Slow
Slow (Reconstructed)
Medium
Medium (Reconstructed)
Fast
Fast (Reconstructed)

The slow sample rate provides at best an inaccurate picture of the signal, while the high sample rate comes much closer to approximating the actual shape of the signal.

So where does TRANSFER fit into this picture? Consider the ENTER and OUTPUT statements with their extensive formatting and conversion capabilities as being like a Rolls-Royce automobile with electric windows, television, liquor cabinet, automatic transmission, and other accessories.

A fast-handshake transfer is then comparable to a Formula I race car, with no windows, manual transmission, one seat, a spine-jolting ride, and that is capable of speeds over 300 kilometers per hour. Its main objective is performance.

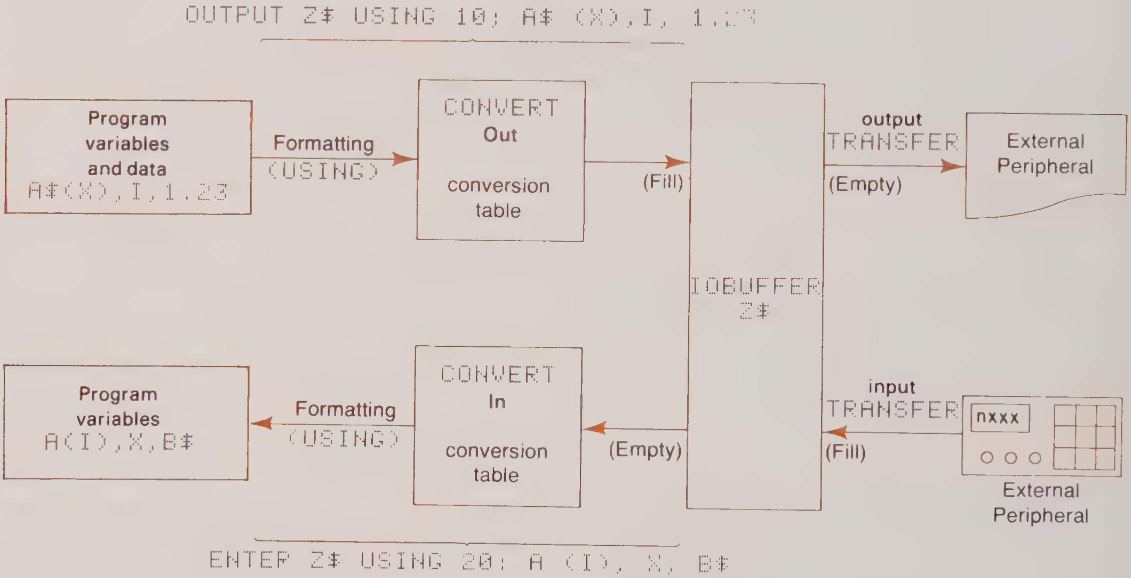
The fast-handshake transfer delivers the highest data transfer rate possible. When a fast-handshake transfer operation is in progress, **all** other activities stop. Even the computer's RESET key is disabled and will have no effect until the transfer completes.

As with the Formula I race car, you pay a price for performance.

Using Buffers

A buffer is a section of read/write memory set aside for the purpose of temporary data storage. It is used to either input data, output data, or both by means of the TRANSFER statement. There is no formatting or data conversion done by TRANSFER, so what is in the buffer is what is sent to the peripheral device. The same holds true for data being input by TRANSFER; it is placed in the buffer exactly as it is received from the device.

To illustrate how output, enter, conversion, formatting, I/O buffers, and transfers work together consider the following diagram.


OUTPUT Z$ USING 10; A$ (X),I, 1.23
Program variables and data
A$(X),I,1.23
Formatting
(USING)
CONVERT Out
conversion table
(Fill)
IOBUFFER Z$
output TRANSFER
(Empty)
External Peripheral
Program variables
A(I),X,B$
Formatting
(USING)
CONVERT In
conversion table
(Empty)
input TRANSFER
(Fill)
nxxx
External Peripheral
ENTER Z$ USING 20: A (I), X, B$

The OUTPUT statement takes data from program variables and does any necessary formatting while placing that data into its ASCII representation. Then, if an output conversion is in effect, the ASCII characters are converted accordingly and placed into the I/O buffer at the position specified by the *fill pointer*. The I/O buffer is *full* when the fill pointer is at the end of the buffer (the string Z$ in this case). You should be aware that default OUTPUT and TRANSFER operations to a buffer place a carriage return and a line feed at the end of the data in the buffer. This means that the buffer should be dimensioned to a length greater by two on the HP-86 and HP-87, when using the default OUTPUT and TRANSFER operations.

The output transfer takes characters from the I/O buffer at the position specified by the empty pointer. These characters are sent to the specified interface and its associated peripheral. This is done either by interrupt or fast-handshake as specified by the programmer. When the transfer completes, the interface's end-of-line character sequence is sent.

The input transfer accepts characters from the specified interface (and its associated peripheral) and places them into the I/O buffer at the position specified by the fill pointer. Again, this is done either by interrupt or fast handshake as specified by the programmer.

The `ENTER` statement takes characters from the I/O buffer at the position specified by the empty pointer. If an input conversion is in effect, these characters are converted accordingly, formatted as necessary, and changed into the proper internal representation for the program variables. If you are entering data from an active buffer, errors can be avoided by using the form `ENTER Z$ USING "#,#K";A$`.

This form of `ENTER` removes the requirement for statement and variable terminators, which may not be in the buffer yet.

The Pointers

On the HP-85, when a string variable is first designated as an I/O buffer (by executing an `IOBUFFER` statement), its dimensioned length is effectively reduced by eight characters. This is to provide room for four "pointers." There is a fill pointer, an empty pointer, an active-out select code, and an active-in select code.* On the HP-86 and HP-87 the buffer length is equal to the dimensioned length.

The *fill pointer* first equals zero (0). This pointer always contains the same value as that returned by the `LEN` function for the string variable. Placing a character into the buffer goes as follows: 1) increment the fill pointer, 2) store the character. This operation is *automatically* handled by the `OUTPUT` (fill pointer equals `LEN` function) statement and also by any string variable assignment operations such as `Z$=Z$&A$`. You do not normally need to assign values to the fill pointer.

The *empty pointer* first equals one (1). Taking a character from the buffer goes as follows: 1) read the character, 2) increment the empty pointer. This operation is performed automatically by the `ENTER` statement.

A buffer is *full* on the HP-83/85 when the fill pointer equals the string's dimensioned length minus 8. A string with a dimensioned length of 8 would not be a very useful buffer, as it would be full and empty at the same time, without any data being placed in it at all! Any `OUTPUT` operation to a full buffer will result in an error. On the HP-87 the length declared is the same as the usable length.

A buffer is *empty* when the empty pointer equals the fill pointer plus one. This has no relationship to the dimensioned length of the string. When a buffer is emptied, the full pointer is reset back to 0 and the empty pointer is reset back to 1. This is exactly the same effect as executing the `IOBUFFER` statement, except that the `IOBUFFER` statement also initializes (destroys) conversion table pointers.

---

*On the HP-87, additional memory is allocated for the pointers (additional to the length that the buffer is declared to be). Because of this, on the HP-87, the number of I/O buffers is limited to ten. Declaring more than 10 buffers will result in an error and abortion of the program at that point.

Buffer Activity

When a `TRANSFER` statement is executed, the specified buffer is then an *active* buffer. The buffer may be *active-out*, *active-in*, or both, depending upon the direction(s) of the transfer(s). The buffer is assigned an *active-in select code* when an input `TRANSFER` statement is executed. An *active-out select code* is assigned when an output `TRANSFER` statement is executed. For example when the following interrupt transfer to select code 6 is executed—

```
TRANSFER Z$ TO 6 INTR
```

the active-out select code equals 6.

A buffer is made *inactive* when the transfer completes. This is a direction-specific inactive state; that is, buffer may be active-out but inactive-in, or vice-versa. When an input transfer completes, the buffer's active-in select code is set to zero (0). Similarly, when an output transfer completes, the buffer's active-out select code is set to zero (0).

Buffer Status and Control

The four buffer pointers can be checked by means of the `STATUS` statement. The four I/O buffer status registers are as follows:

Status Register	Default Value	Register Function	Statement used to read register value of buffer Z$
SR0	1	Buffer empty pointer	`STATUS Z$,0,T0`
SR1	0	Buffer fill pointer	`STATUS Z$,1;T1`
SR2	0	Active-in select code	`STATUS Z$,2;T2`
SR3	0	Active-out select code	`STATUS Z$,3;T3`

These registers may be read at any time on an active or inactive buffer, but attempting to read the status of a non-buffer string variable (that is, if no `IOBUFFER` statement has been executed for that string variable) results in an `ERROR`.

An example of using buffer status registers to control program flow follows. In the example, a string variable is dimensioned to 88 characters (to allow for 80 characters of data after becoming a buffer on the HP-85) and declared as an I/O buffer. Data is output to the buffer, a transfer out to an HP-IB printer is initiated, then the buffer's active-out select code is checked to determeine when the transfer has completed. The program ends when the transfer completes. The buffer may be dimensioned to 80 characters on the HP-86 or HP-87.

```
 10 DIM Z$[88]
 20 IOBUFFER Z$ ! Usable size of Z$ is 80 (88 on the HP-86/87)
 30 FOR I=0 TO 1 STEP .1 ! EOL is suppressed
 40 OUTPUT Z$ USING "#,D.DDD,X" ; SIN(I)
 50 NEXT I
 60 ! Show values of buffer registers
 70 STATUS Z$,0 ; T0,T1,T2,T3
 80 PRINT "Registers before TRANSFER ",T0,T1,T2,T3
 90 TRANSFER Z$ TO 701 INTR
100 STATUS Z$,0 ; T0,T1,T2,T3
110 DISP T0,T1,T2,T3
120 IF T3<>0 THEN GOTO 100
130 PRINT "Transfer completed"
140 END
```

The variable T0 shows how many characters have been taken from the buffer. T1 shows how many characters total are in the buffer, and T3 indicates select code activity.

For another example, assume device `X` is to send three numeric values followed by a carriage return, line feed. The following program displays the buffer registers until the transfer completes. An `ENTER` is then executed to take the values out of the buffer, which are then printed.

```
 10 ! Our device X is at HP-IB address 721
 20 DIM Z$[88]
 30 IOBUFFER Z$
 40 TRANSFER 721 TO Z$ INTR ; DELIM 10
 50 ! Note that 10 is the decimal value for line-feed
 60 STATUS Z$,0 ; T0,T1,T2,T3
 70 DISP T0,T1,T2,T3
 80 IF T2<>0 THEN GOTO 60
 90 ENTER Z$ ; X,Y,Z
100 DISP X,Y,Z
110 END
```

If device `X` is a very slow device, you could "watch" characters come into the buffer by means of variable `T1`. The variable `T0` is not altered until the `ENTER` is executed, and `T2` reflects the `TRANSFER` statement activity. When the `TRANSFER` is initiated, `T2=7`; when the `TRANSFER` completes, `T2=0`.

The buffer empty pointer and the buffer fill pointer can be assigned new values by using the `CONTROL` statement. This gives you the capability of sending the same data over and over again without having to re-compute the data, for example. The following table shows these registers and how they are accessed:

Control Register	Default Value	Register Function	Statement used to write register value of buffer Z$
CR0	1	Buffer empty pointer	`CONTROL Z$,0;V0`
CR1	0	Buffer fill pointer	`CONTROL Z$,1;V1`

These buffer registers may be written to at any time, but attempting to write to a control register of a non-buffer string variable (before an `IOBUFFER` statement has been executed for the string variable) results in an error.

In the following example, 360 values are computed for `SIN(X)` to represent one complete cycle of a sine wave. These values are output to the buffer `Z$` and subsequently sent to device X, a digital-to-analog converter, by a fast-handshake transfer.

When the transfer completes (T2 = 0), the buffer empty pointer is automatically reset to 1, the fill pointer is set equal to 360, and the transfer restarted. This continues indefinitely until the PAUSE key is pressed to stop the program. The effect of this program is to produce a near-continuous sine wave from device X. (Details of device X are purposefully left out here to avoid confusing the issue.)

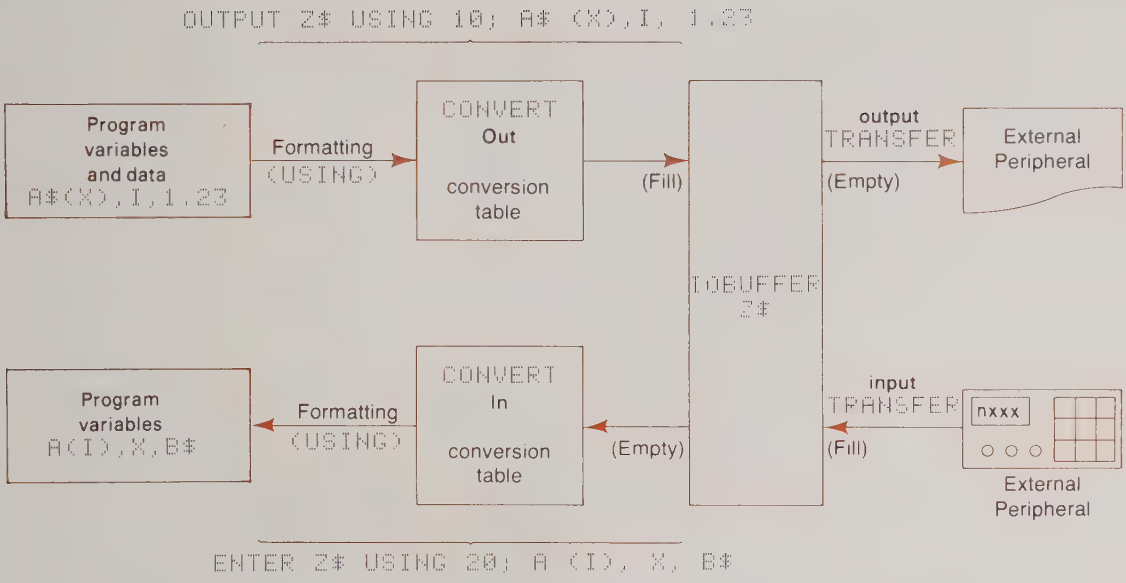
```
 10 ! Make room for 360 eight-bit values plus pointers
 20 DIM Z$[368]! 360 values are adequate for the HP\86/87.
 30 IOBUFFER Z$
 40 DEG
 50 FOR I=0 TO 359
 60 OUTPUT Z$ USING "#,B" ; SIN(I)*255
 70 DISP I;@ NEXT I
 80 TRANSFER Z$ TO 5 FHS
 90 CONTROL Z$,1 ; 360
100 GOTO 80
110 END
```

Lines 10-40 merely set up the buffer and place the computer in degrees mode. Lines 50-70 fill the buffer with the 8-bit value for each of 360 degrees (one complete sine wave), and line 80 starts the fast-handshake transfer. Line 90 is executed when the transfer is complete. The buffer fill pointer is set back to 360 (so it looks full again), and the transfer is restarted.

By exercising control over the buffer empty and fill pointer, it is possible to retransmit data, transmit any portion of the data in the buffer, to write data into any section of the buffer, read data out of any section of the buffer, etc. These operations may not be ones that you need to use in your application, but the flexibility they provide you could make feasible certain I/O operations not possible through any other means.

Data TRANSFERS

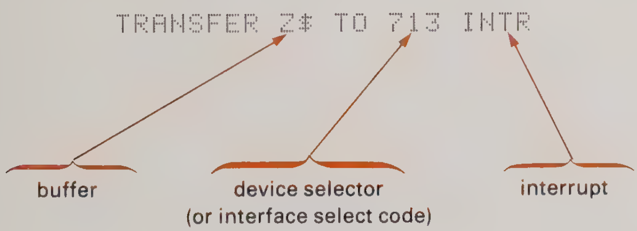
The `TRANSFER` statement has been mentioned several times up to this point. In combination with the `IOBUFFER`, it provides you with unmatched flexibility in tailoring and optimizing a program to exchange data with one or more peripheral devices. The diagram below shows the relationship of the transfer with the I/O buffer, conversion tables, and program variables.


OUTPUT Z$ USING 10; A$ (X),I, 1.23
Program variables and data A$(X),I,1.23
Formatting (USING)
CONVERT Out conversion table
(Fill)
IOBUFFER Z$
output TRANSFER
(Empty)
External Peripheral
Program variables A(I),X,B$
Formatting (USING)
CONVERT In conversion table
(Empty)
input TRANSFER
(Fill)
nxxx
External Peripheral
ENTER Z$ USING 20; A (I), X, B$

The transfer itself is the easiest section of the entire picture to understand. Simply stated, an output transfer takes characters, or bytes, out of the buffer from the position specified by the buffer empty pointer and sends them to the external peripheral. Conversely, the input transfer takes characters from the external peripheral and places them in the buffer at the position specified by the buffer fill pointer.

Output TRANSFER

For an output transfer, you merely specify whether an interrupt or a fast-handshake transfer is to be performed. For example, to specify an interrupt output transfer, the following statement could be used:


TRANSFER Z$ TO 713 INTR
buffer
device selector (or interface select code)
interrupt

The operation of an interrupt output transfer is as follows:

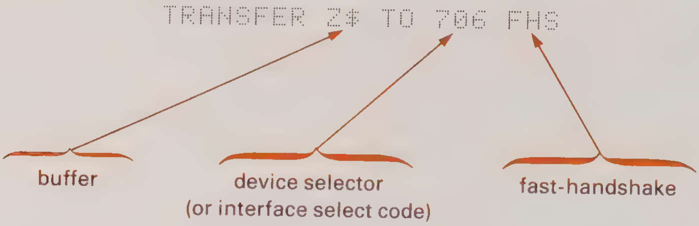
- When the statement is executed, the interface involved (in this case, device 13 on an HP-IB Interface at select code 7) is automatically enabled to *interrupt** the computer when ready to accept a new character.
- Thereafter, each time the interface interrupts, the computer temporarily suspends program execution long enough to move the next character from the buffer to the interface. (It is then the interface's responsibility to see that the new character is properly sent to the peripheral.)

---

*This is a hardware-level interrupt, and is distinct from the software-level end-of-line branch discussed in Section 6.

- When the buffer is finally emptied (described in the Using Buffers section), the computer disables (or turns off) further interrupts from the interface. The transfer is now essentially complete, although the interface may still be sending out the specified end-of-line character sequence (normally a carriage return/line feed: see the appropriate interface owner's manual).
- When the transfer is terminated, a check is made for a user-defined end-of-transfer branch. If one is defined, then the branch is taken upon completion of the current program line. This is explained more thoroughly in the section End-of-Line Branching.

A fast-handshake output transfer is specified as—



```
TRANSFER Z$ TO 706 FHS
```

The operation of a fast handshake output transfer is significantly different:

- The computer suspends program execution and dedicates itself to the task of moving all the characters in the buffer to the interface. The computer *totally* devotes itself to this task until the buffer is emptied. No other interrupts are allowed, not even a RESET! This means that once a fast-handshake transfer begins the computer will see it through to completion, and all other operations are ignored until that time.
- When the buffer is empty, the transfer is essentially finished. (The interface may still be in the process of sending the end-of-line character sequence.) If a user-defined end-of-transfer branch is specified, the branch is taken upon completion of the current program line.

Input TRANSFER

The input transfer is essentially the same as the output transfer, except that data is being moved from the external peripheral to the computer. However, in addition to specifying whether an interrupt or fast-handshake transfer is to be performed, you may also specify the terminating condition(s) for the transfer. The terminating conditions for *both* interrupt and fast-handshake transfers are:

- `COUNT`—the number of characters or bytes of data to be input before the transfer is considered complete. Use `COUNT` when the number of characters or bytes being sent from the peripheral device is known.
- `EOI`—an interface-specific terminating condition. In the case of the HP-IB Interface, some devices set the EOI control line when sending the last byte of data indicating end-of-data. This terminates the transfer if `EOI` is specified.
- *Default termination*—a full buffer termination. An input transfer terminates with the first occurrence of any specified terminating condition or a full buffer. (Obviously, if there is no room left in the I/O buffer, the transfer cannot continue.) If no other terminating condition has been specified, an input transfer will terminate when the I/O buffer becomes full.

Additional terminating conditions for interrupt transfers *only* are:

- `DELIM`—the numeric value of the character or byte of data that indicates all data has been sent. Use `DELIM` when the number of characters or bytes being sent from the peripheral device is either not known or varies for some reason. The peripheral device should not send the `DELIM` character as part of the data being sent, or else the transfer will be terminated prematurely!
- *Interface termination*—an interface specific termination. Certain interfaces allow you to specify additional terminating conditions by writing to the interface control registers. These conditions may be either receipt of user-specified characters or line status, so consult the appropriate interface programming section to determine which, if any, conditions your interface supports.

Programming with TRANSFERS

Every transfer operation requires an I/O buffer, which in turn requires a dimensioned string variable of adequate size. *After* executing the `IOBUFFER` statement, any `CONVERT` statement(s) for that buffer may be executed. This is because the `IOBUFFER` statement initializes all the buffer pointers, including those to any conversion tables.

If there is any necessary interface initialization to do, it may be done anytime before executing the transfer. If your program uses end-of-line branching (`ON EOT`: see the End-of-Line Branching section), the appropriate statement(s) should be executed before the `TRANSFER` statement.

An input transfer may be executed at any point after the setup sequence described above. An output transfer, however, requires data in the buffer before anything can be sent. Therefore, an output or a string assignment operation needs to be performed before attempting to execute the output transfer.

It has been said that either an output or a string assignment operation needs to be performed before executing an output transfer so that there is data available in the buffer. This is not entirely true. Nor is it necessary to have all the data in the buffer before initiating an output interrupt transfer. Also, it is not necessary to wait for an input interrupt transfer to complete before reading data from the buffer. (Fast-handshake transfers are totally sequential operations and do not apply to the following discussion.)

If the external device is sufficiently slower than the computational speed of the computer, it is possible to initiate an output transfer with the buffer only partially filled. Then, as more data becomes available to send to the device, the new data can be output (for example) to the buffer while the transfer is still in progress. This process can go on indefinitely until either: 1) all data has been sent or, 2) the buffer is filled (then the user's program must wait for more buffer space) or, 3) the buffer is emptied (then the transfer terminates and it must be restarted).

Case 1 above is straightforward and needs no explanation. Case 2 is determined by the buffer status, and the program can wait for more buffer space by monitoring buffer status (with the `STATUS` statement). The occurrence of case 3 indicates that the peripheral device is actually faster than the computer in this instance. (This may be due to excessive computation being performed or some other circumstance causing a delay in the program.) An end-of-line branch or a status check on the buffer indicates a transfer completion, and the decision to continue the transfer can be made at that time. The following example program illustrates how these conditions might be dealt with:

```
10 ! This program shows how you might check for cases 1,
20 ! 2,and 3. The buffer size is determined empirically
30 ! to optimize memory usage with transfer rate.
```

```
 40 DEG @ DIM B$[160]
 50 DEF FNA(X) = INT(SIN(X)*35+36.5)
 51 CONTROL 7,16 ; 0 ! This turns off CR/LF from the
    interface by specifying zero EOL sequence characters
 60 IOBUFFER B$
 70 !
 80 FOR X=0 TO 359 STEP 10 ! The For-Next loop handles
    case 1
 90 STATUS B$,0 ; S1,S2,S3,S4
100 ! Test for buffer empty before putting in data (case
    2)
101 Z=FNA(X)
110 IF S1+Z>145 THEN GOTO 90
115 I$=VAL$(Z)&"X,A"
120 OUTPUT B$ USING I$ ; "*"
130 ! Test the active-out select code before starting the
    transfer
140 IF S4=0 THEN TRANSFER B$ TO 720 INTR ! (case3)
150 NEXT X
160 !
170 GOTO 80
180 END
```

Section 6

End-of-Line Branching

Some Background on Interrupts

If you've never heard the term "interrupt" with regards to a computer before, then a simple way to think of it is like a telephone ringing while you are working. It is a means of diverting your attention from whatever it is you are doing. "Servicing" an interrupt is similar to the act of going over to answer the ringing telephone. When the telephone business is completed, you typically resume the "interrupted" activity where you left off.

If you have a switch that can disconnect the telephone bell (so it can't ring), then you can disable or enable that interrupt by the setting of the switch. The computer has essentially the same facility.

In addition, the computer has two types of interrupts to deal with: low-level, or hardware interrupts; and high-level, or software interrupts. In general, the hardware interrupt is used by the computer for its own purposes, and is transparent to the user. Hardware interrupts make possible such things as the `INTR` type of `TRANSFER`, where program execution and I/O are concurrent operations. However, external (and internal) events can also trigger hardware interrupts that require specialized service routines to deal with them. Because there is such a wide variety of interrupt causes possible, it is necessary to make provisions for the programmer to custom tailor service routines for those interrupts specific to his system. This provision is the software interrupt, or end-of-line branch.

Imagine a "ghost" `IF` *<event>* `THEN GOSUB` *<line #>* statement appended to the end of each program line, like the ones below:

```
50 PRINT "Result=";X @IF <event=true> THEN GOSUB <routine>
60 X=X+Y @IF <event=true> THEN GOSUB <routine>
```

In effect, at the end of each program line, a check is made for the *event*, which may be an error, a hardware interrupt, a select code timeout, a transfer completion, a special function key-press, or a timer timeout. (End-of-line service for three of the above events is already provided for in the HP Series 80 mainframe: errors, special function keys, and timers.) When one of the events occurs, a special portion of the program can be executed to deal with that event.

The programmer also has the ability to disable or enable the end-of-line service facility for each event, as will be shown in the next section on programming for end-of-line branches.

End-of-Line Branch Programming

Introduction

This section shows you how to program service routines to deal with end-of-line branches for I/O related events. At the end of this section is a sub-section entitled "Interactions and Permutations." The discussion there deals with problems and questions you may encounter such as `GOSUB` vs. `GOTO`.

Interface Interrupts

Each interface (HP-IB, Serial, BCD, etc.) has a control register that allows you to specify an interrupting condition. This is control register CR1, the Interrupt Mask register. When you set a bit in that register, you *enable* the interface to *interrupt* the computer when the event corresponding to that bit occurs. The interrupt event is determined by referring to the programming section for the particular interface. For example, the HP-IB Interface provides for an interrupt on SRQ, or Service Request. The SRQ interrupt is enabled by setting bit 3 (decimal value 8) of control register CR1. To enable the SRQ interrupt for the HP-IB Interface at select code 7, execute

```
ENABLE INTR 7;8
```

This is equivalent to

```
CONTROL 7,1;8
```

and both statements set bit 3 of control register CR1 on select code 7.

Suppose (for simplicity) that only one device is connected to the HP-IB Interface, and that device asserts SRQ only to indicate it is ready to send a numeric value to the computer. The statement we'll use to read this value is

```
ENTER 701;V1
```

This is the extent of our example service routine, which will take the form of a subroutine in the program.

Once the computer has read the new value from the device, it must again enable the SRQ interrupt so that another value can be read when the device is again ready. The service routine begins to take shape:

```
1000 ! SRQ Service Routine for select code 7
1010 STATUS 7,1 ; A ! Always read interrupt cause register
1020 ENTER 701 ; V1
1030 ! Re-enable interrupt and return on same line
1040 ENABLE INTR 7;8 @ RETURN
```

Note that the `ENABLE` and the `RETURN` are on the same program line. There are some occasions where this is a necessary practice, and this is discussed under "Interactions and Permutations."

It is necessary to specify *where* the service routine for an event is, and also whether the routine is a subroutine (`GOSUB`) or just a program segment (`GOTO`). This is accomplished by the `ON INTR` statement. Note that the computer must know where to go and what to do before an interrupt occurs, or else it will be forced to ignore it. Therefore, the `ON INTR` statement should be executed before the `ENABLE INTR` statement, as shown in the example below.

```
10 ON INTR 7 GOSUB 1000
20 OUTPUT 722 ;"F1R7T2T3D1"
30 V1=X=0
40 LOCAL 722
50 ENABLE INTR 7;8
```

```
  60 ! The following lines are a dummy program
  70 X=X+1 @ DISP X,V1
  80 WAIT 100 @ GOTO 70
     :
1000 ! SRQ Service Routine for select code 7
1010 STATUS 7,1 ; A
1020 ENTER 722 ; V1
1030 ENABLE INTR 7;8 @ RETURN ! This must be done to reset
     interrupt
1040 END
```

Line 10 specifies the location (line 1000) and type (`GOSUB`, not `GOTO`) of the service routine for select code 7. Line 20 initializes the HP 3455A Digital Voltmeter to take readings whenever it is triggered. The `LOCAL` statement of line 40 allows the DVM to be triggered manually, and the `ENABLE` statement enables the interface for SRQ interrupt (bit 3 of the interrupt mask is set). Lines 70 and 80 merely display an incrementing counter and the last reading taken from the DVM.

Each time an SRQ is received, program execution of lines 70 and 80 is suspended, and the subroutine at line 1000 is executed. The `ENTER` is performed, which takes a new value for `V1`. The SRQ interrupt condition is again enabled in line 1030, and the `RETURN` allows the program display loop to resume. However, the value for `V1` that is now displayed is the new value just read by the service routine.

What would happen if the `ENABLE INTR` on line 1030 were removed?

Simple. The first SRQ would be serviced as described above, and the program loop then resumed. However, no more SRQ interrupts would be detected, even if the HP-IB device were frantically sending the SRQ message to the interface! The interface *ignores* the event—regardless of its importance, until an `ENABLE INTR` statement is performed to again enable the interface to be able to interrupt the computer. Also, the `ENABLE INTR` statement sets the status register to the current state.

To disable or cancel the end-of-line branch condition set up by `ON INTR`, simply execute an `OFF INTR` statement for the appropriate select code. No more branches will be taken until another `ON INTR` statement is executed.

It is good programming practice to always read the interrupt cause register. Note, however, that the execution of a `STATUS` statement clears the status register so it cannot meaningfully be read again immediately. It may also indicate more than one interrupt.

Timeouts

It is possible for an external device to exhibit symptoms of intermittency or disconnection from time to time. This "falling asleep" is generally caused by an electronic malfunction or some other unusual condition, such as being switched off. It is not as important to focus here on the possible *causes* of device failure so much as the possible *effects*.

When a device fails, for whatever reason, the immediate effect on the computer is that it is no longer handshaking data to or from the computer. (Handshaking is a means of reliably transferring data between two devices. The sender makes data available, signals that data is ready, and the receiver accepts the data and signals that it has taken the data. This protocol is called handshake.) If an `ENTER`, `SEND` or `OUTPUT` operation is in progress, a loss of handshake means no data can be transferred and the operation cannot complete. If the operation does not complete, the program "hangs" until the operator becomes aware that something has gone wrong. This is unacceptable for most I/O systems, and especially those where unattended operation is frequent.

The `SET TIMEOUT` statement gives you the capability of establishing a maximum time period for the computer to wait on an interface to handshake data. If an interface exceeds the `SET TIMEOUT` limit specified, two alternative courses of action may be selected. One method simply aborts the I/O operation in progress and continues program execution at the next line. The other method executes a timeout service routine after the I/O operation is aborted.

The following example shows the simpler form of `SET TIMEOUT`, where no timeout service routine is specified:

```
10 SET TIMEOUT 7;10000
20 OUTPUT 706 ;"Simple test data"
30 DISP "Now at line 30"
40 END
```

If device 706 stops responding to handshake before the output operation is complete, then after 10 seconds line 30 will be executed. (Line 30 would also be executed if the output were successful, obviously.)

A more sophisticated method for dealing with a timeout condition is to execute a timeout service routine when an I/O operation is aborted due to a timeout. A separate service routine may be programmed for each select code to deal with the specific device or conditions for that select code. The `ON TIMEOUT` statement provides this capability, as shown below:

```
  10 SET TIMEOUT 7;2000
  20 SET TIMEOUT 5;4000
  30 ON TIMEOUT 7 GOSUB 1000
  40 ON TIMEOUT 5 GOSUB 1100
  50 ENTER 706 ; V1,X
  60 OUTPUT 5 ;V1,X
  70 GOTO 50
     ⋮
1000 ! Timeout service for select code 7
1010 PRINT "HP-IB Timeout"
1020 RESET 7 ! Attempt to recover
1030 RETURN
1100 ! Timeout service for select code 5
1110 PRINT "Select code 5 failure"
1120 RESET 5
1130 RETURN
```

Obviously, in a dedicated system, more significant action could be taken than simply printing a message, aborting the operation with a `RESET`, and returning to the program. For example, a flag could be set by the timeout routine which would be checked before attempting an I/O operation to that select code. This flag might indicate printer out of paper or tape reader at end of tape, which would make I/O to such a device a useless endeavor. It might also be the only means available of determining when an operation has completed. The following example shows how a flag might be used:

```
10 SET TIMEOUT 6;1500
20 ON TIMEOUT 6 GOSUB 1000
21 ! Clear timeout flag (F1)
30 F1=0 @ DIM V(50)
40 FOR I=1 TO 50
50 IF F1<>0 THEN GOTO 80 ! Test timeout flag before
   ENTER
```

```
  60 ENTER 6 ; V(I)
  70 NEXT I
  80 PRINT I;" Values entered" @ STOP
     :
1000 DISP "Timeout : end of data"
1010 F1=1 ! Set timeout flag
1020 RETURN
1030 END
```

When a timeout occurs, the flag F1 is set equal to 1, and the `IF` statement of line 50 causes the `ENTER` statement of line 60 to be skipped. The loop is exited and the program continues. (In the example, only a `PRINT` and `STOP` are shown for simplicity.) In this case, the cause for the timeout is not device failure, but rather the end of data to be received.

As with `ON INTR`, there is a corresponding disable statement for timeout end-of-line branches. The `OFF TIMEOUT` statement can be used to cancel end-of-line service for a specified interface timeout. The `SET TIMEOUT` statement with a time limit of 0 can also be used, which sets an (almost) infinite timeout limit for the interface. For more information regarding syntax rules, refer to the alphabetical syntax listing section of this manual.

Transfer Terminations

There are two means of determining when a transfer operation has completed. Both are straightforward, but using an end-of-line branch service routine allows the greatest flexibility of program design.

As is mentioned in "Using Buffers", buffer status registers SR2 and SR3 indicate input and output activity, respectively. When the appropriate register becomes equal to 0, that transfer has terminated. However, this requires rechecking the status register until the transfer terminates, which inhibits program execution. If there is other work to be done, a fairly convoluted program logic will be the result of trying to monitor transfer status and also getting the other work done. There is a simpler way.

When an `ON EOT` (On End Of Transfer) statement is executed, an end-of-line branch is enabled for transfer termination for the specified interface. Then you can program the operations necessary to deal with the transfer termination into the `ON EOT` service routine. These operations might include placing new values into the buffer and re-initiating the transfer (for output) or taking values from the buffer and placing them into program variables (for input).

To illustrate how the `ON EOT` statement might appear in a program, consider the following example. An HP 3437A Voltmeter is programmed to take a reading every .9 seconds. An interrupt transfer is set up for 100 readings of seven characters each, plus a final carriage return/line feed and the eight extra bytes needed for pointers.

When the transfer completes, the service routine at line 1000 is executed which saves the readings and initiates a new transfer.

```
10 ! Transfer termination service routine example
20 ON EOT 7 GOSUB 1000
30 OUTPUT 724 ;"D.9SN100SE0SR3T1F1"
40 I=1 @ X=0 @ DIM D$[710],D(100)
50 IOBUFFER D$
60 ! This program assumes 10 data files are already
```

```
  70 ! created, and that the file names are in the next
  80 ! data statement.
  90 DATA data1,data2,data3,data4,data5,data6,data7,data8,
     data9,data10
 100 TRANSFER 724 TO D$ INTR
 110 X=X+1 ! Simple "work" loop (dummy)
 120 DISP X
 130 GOTO 110
     :
1000 ! EOT service routine
1010 ! Read next file name and open file on tape.
1020 READ F$@ ASSIGN# 1 TO F$
1030 ! Take values from buffer
1040 FOR N=1 TO 100
1050 ENTER U$ ; D(N)
1060 NEXT N
1070 ! Now initiate a new TRANSFER
1080 TRANSFER 724 TO D$ INTR
1090 ! Save data on tape. NOTE: this temporarily halts the
     transfer!
1100 PRINT# 1 ; D() @ ASSIGN# 1 TO *
1110 ! See if all 10 data files have been saved.
1120 I=I+1 @ IF I>10 THEN GOTO 1140
1130 RETURN
1140 ! If done, then halt I/O and end program.
1145 ! Note that ABORTIO causes an EOT branch, so disable
     it.
1150 OFF EOT 7 @ ABORTIO 7
1160 PRINT "All data saved"
1170 END
```

The program logic associated with the transfer and the EOT service routine is quite simple (shaded program lines). The extra programming shown is included as one example of how the EOT service might be used for data-save operations to tape. The EOT service routine might just as well have sent the collected data to a remote central computer (with another `TRANSFER`, of course!).

End-of-transfer service can be cancelled by execution of an `OFF EOT` statement for the desired select code. For further details about these statements, refer to the alphabetical syntax listing in this manual.

Interactions and Permutations

This section is written so that you may be able to better predict operation of those programs that utilize end-of-line branching.

End-of-line service occurs in a specific order. That is, if more than one end-of-line branch is pending at the end of a program line, one of the branches will be taken before the other. The following table lists the types of end-of-line branches and the select codes, and gives the precedence order for combinations of branch type and select code.

Branch Precedence Table

Branch Type	Select Code							
	3	4	5	6	7	8	9	10
ON ERROR	1							
ON INTR	2	3	4	5	6	7	8	9
ON TIMEOUT	10	11	12	13	14	15	16	17
ON EOT	18	19	20	21	22	23	24	25
ON KEY	26							
ON TIMER	27							

For example, a pending `ON INTR` branch for select code 5 would be taken before a pending `ON INTR` branch for select code 9. *Any* pending `ON EOT` branch would be taken *after* any pending `ON TIMEOUT` branch, regardless of select codes.

You should take note that the term *precedence* is used here, not *priority*. This means that when the computer is executing one service routine, other service routines are not implicitly locked out. (In a priority system, any service routine having a lower priority than the routine currently being executed will not receive control until the current routine completes.) If two end-of-line branches are pending on the HP Series 80 computer, the one having precedence is executed first. However, after the first line of that service routine executes, the still-pending end-of-line branch to the second service routine is taken!

The only way the first routine can guarantee uninterrupted execution is to disable the other routine's end-of-line branch with its first line. An example will help show this:

```
 10 ON INTR 7 GOSUB 100
 20 ON INTR 9 GOSUB 200
 30 ENABLE INTR 7;8
 40 ENABLE INTR 9;64
    :
 70 ! Program body goes here
    :
100 OFF INTR 9 ! Now select code 9 cannot get EOL service.
    :
130 ! This routine would do necessary service functions
140 ! for select code 7.
    :
180 ! Now re-enable EOL service for select code 9
190 ON INTR 9 GOSUB 200 @ ENABLE INTR 7;8 @ RETURN
200 OFF INTR 7 ! Now select code 7 cannot get EOL service.
    :
240 ! This routine would do necessary service functions
250 ! for select code 9.
    :
280 ! Now re-enable EOL service for select code 7
290 ON INTR 7 GOSUB 100 @ ENABLE INTR 9;64 @ RETURN
300 END
```

Assume that while line 50 is executing, both interrupts occur. Both end-of-line branches are pending when line 50 completes. Select code 7's service routine has precedence, and so line 100 is executed. Line 100 disables select code 9's service routine, so the end-of-line branch that would have occurred is not taken. Line 190, the last line of the select code 7 service routine, re-enables the select code 9 service routine and when the `RETURN` is executed, line 200 is then executed. Note that the `OFF INTR` does not *eliminate* the pending end-of-line branch—it just defers the branch until an `ON INTR` is executed later.

Events such as interrupts, timeouts, transfer terminations and errors are "remembered" indefinitely, until a `RESET` or `STOP` occurs. Therefore, once an event has occurred and remains unserviced, execution of an `ON INTR`, `ON TIMEOUT` or `ON EOT` results in an immediate end-of-line branch.

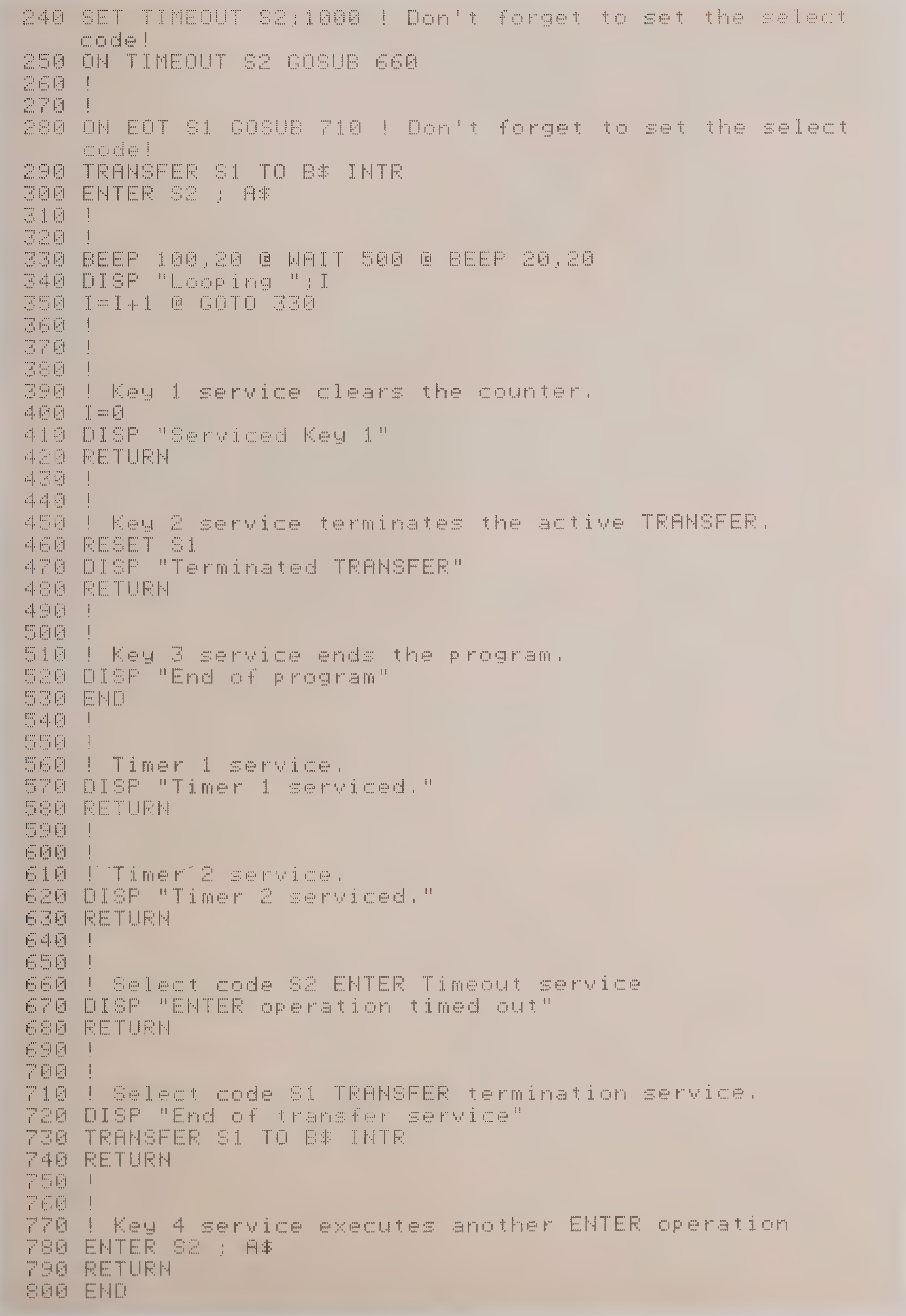
The *type* of branch taken can affect program operation. The most predictable program execution occurs when `GOSUB` end-of-line branches are taken. Use `GOTO` when the program is simple and only one or two end-of-line branches are expected. One very simple example of this would be:

```
ON INTR 7 GOTO 9999          (99999 for the HP-87)
ENABLE INTR 7;128
```

These two statements direct program execution to line 9999 (presumably an `END` statement) when an Interface Clear on HP-IB occurs. This kind of "bail-out" provision can be useful when developing certain types of programs such as ones using keyboard masking (see section 7). Then, when something goes wrong you can halt the program by asserting IFC on the interface bus with another controller or a bus analyzer.

The following example is presented to show you how some of the ON-event branches interact. The listing and results are shown with a brief explanation, but you will learn more from it if you experiment on your own. Try setting timers, timeouts, and keys up and then watch it work (or not work). Press the special function key while the computer is waiting, then watch the display.

```
 10 ! This program serves to illustrate end-of-line
    branching.
 20 ! Substitute YOUR select codes and device numbers
 30 ! on lines 80 and 90 to make this run successfully.
 40 ! Remember that an ENTER and an input TRANSFER cannot
 50 ! occur simultaneously on the same select code.
 60 !
 70 !
 80 S1=10 ! This is the TRANSFER select code assignment.
 90 S2=705 ! This is the ENTER select code assignment.
100 I=0 @ DIM A$[80],B$[88]
110 IOBUFFER B$
120 !
130 !
140 ON KEY# 1 GOSUB 390 ! Key 1 clears the counter
150 ON KEY# 2 GOSUB 450 ! Key 2 terminates the TRANSFER
160 ON KEY# 3 GOSUB 510 ! Key 3 ends the program
170 ON KEY# 4 GOSUB 770 ! Key 4 starts another ENTER
180 !
190 !
200 ON TIMER# 1,1300 GOSUB 560
210 ON TIMER# 2,1400 GOSUB 610
220 !
230 !
```



```
240 SET TIMEOUT S2;1000 ! Don't forget to set the select
    code!
250 ON TIMEOUT S2 GOSUB 660
260 !
270 !
280 ON EOT S1 GOSUB 710 ! Don't forget to set the select
    code!
290 TRANSFER S1 TO B$ INTR
300 ENTER S2 ; A$
310 !
320 !
330 BEEP 100,20 @ WAIT 500 @ BEEP 20,20
340 DISP "Looping ";I
350 I=I+1 @ GOTO 330
360 !
370 !
380 !
390 ! Key 1 service clears the counter.
400 I=0
410 DISP "Serviced Key 1"
420 RETURN
430 !
440 !
450 ! Key 2 service terminates the active TRANSFER.
460 RESET S1
470 DISP "Terminated TRANSFER"
480 RETURN
490 !
500 !
510 ! Key 3 service ends the program.
520 DISP "End of program"
530 END
540 !
550 !
560 ! Timer 1 service.
570 DISP "Timer 1 serviced."
580 RETURN
590 !
600 !
610 ! Timer 2 service.
620 DISP "Timer 2 serviced."
630 RETURN
640 !
650 !
660 ! Select code S2 ENTER Timeout service
670 DISP "ENTER operation timed out"
680 RETURN
690 !
700 !
710 ! Select code S1 TRANSFER termination service.
720 DISP "End of transfer service"
730 TRANSFER S1 TO B$ INTR
740 RETURN
750 !
760 !
770 ! Key 4 service executes another ENTER operation
780 ENTER S2 ; A$
790 RETURN
800 END
```

Section 7

Keyboard Control

Introduction

In certain applications, the operation of a system could be adversely affected by a curious passerby pressing keys which could halt or alter program execution. In other applications the system operator must have access to certain keys, but not others.

Traditionally, the dedicated computer's keyboard was covered by a "mask" of plastic, metal or wood which covered all keys unnecessary to the operation of the system. Cut-outs were provided to allow access to those keys the operator used to control the system. Obviously, the extra cost of designing and manufacturing such a mask added to the cost of a dedicated system.

The I/O ROM provides the keyboard mask in software, which—short of turning the computer off—offers far less chance of user interference than does a mechanical mask. In addition, greater flexibility is possible with the software masks, as is discussed in the following section.

Key Mask Programming

There are three basic modes of operation for the HP Series 80 Personal Computer: halted (or idle), program execution, and keyboard input.

In the idle mode, the computer accepts commands and is available for program modification/entry. This is the mode that the computer normally is in when not executing a program (when first turned on, or after executing a STOP, PAUSE, or END type statement). The idle mode is generally used for program development or debugging, so a keyboard mask for the idle mode is not very useful and is not provided.

In the program execution mode, pressing any key other than the TR/NORM key or an ON KEY defined special function key will halt the program. A keyboard mask can be specified for four classes of keys while in the program execution mode. These classes are:

1. RESET
2. PAUSE
3. Special function keys and KEY LABEL key
4. Other keys (all remaining keys not in classes 1, 2, or 3)

Any or all four classes of keys can be masked out for the program execution mode.

In the keyboard input mode, the computer is temporarily halted awaiting operator response. This is the mode of operation for the INPUT statement. The four classes of key masks for the keyboard input mode are:

1. RESET
2. PAUSE
3. Special function keys and KEY LABEL key
4. Other keys (all remaining keys not in classes 1, 2, or 3)

The `ENABLE KBD` statement takes as a parameter the key mask desired for the appropriate operating mode. The upper four bits of the mask parameter specify the program execution keyboard mask. The lower four bits specify the keyboard input key mask. The bits of the mask parameter are shown below. Setting a bit *enables* the corresponding key(s), while clearing a bit *disables* the key(s). That is, only those keys having a bit set in the `ENABLE KBD` mask will respond.

Bit Number	Decimal Value	Operating Mode	Keys Not Masked
7	128	Program Execution	RESET
6	64		PAUSE
5	32		Special function keys and KEY LABEL
4	16		Other keys
3	8	Keyboard Input	RESET
2	4		PAUSE
1	2		Special function keys and KEY LABEL
0	1		Other keys

The following example illustrates the use of the keyboard mask to disable all keys except the special function keys for program execution. The RESET, PAUSE, and special function keys are masked out for the keyboard input mode. This still allows the operator to respond to an `INPUT` statement, but program execution cannot be affected.

```
  10 ENABLE KBD 1+32 ! Select allowed keys
  20 REMOTE 706,710 @ LOCAL LOCKOUT 7
  30 ON KEY# 1 GOTO 1000
  40 ON KEY# 2 GOTO 2000
  50 ON KEY# 3 GOTO 3000
     :
 100 PRINT "Select operating mode"
     :
1000 DISP "Enter date" @ INPUT M,D,Y
1010 ! and etc.
```

The operator can select the desired program section by pressing the appropriate special function key—but only the special function keys will respond to being pressed. When the operator presses special function key #1, the program branches to line 1000. When the `INPUT` statement of line 1000 is executed, the special function keys no longer respond but the numeric keys do, allowing the operator to set the time as requested. When the program continues on from the `INPUT`, once again all keys except special function keys are disabled.

The following example gives the operator a ten second time span in which he can pause or reset the computer if necessary:

```
10 ! Line 50 enables the operator to pause or reset the
20 ! program whenever an INPUT statement is executed.
30 ! Line 100 exists specifically to allow the operator
40 ! to halt the program if necessary.
50 ENABLE KBD 12
60 !
70 !
80 !
```

```
 90 BEEP @ DISP "You have 10 sec to PAUSE or RESET the
    computer"
100 ON TIMER# 1,10000 GOTO 110 @ INPUT X9$
110 OFF TIMER# 1 @ DISP "Computer beginning automatic
    operations"
120 !
130 ! Now the operator cannot halt the program!
140 !
150 !
```

Section 8

Direct Interface Communication

Introduction

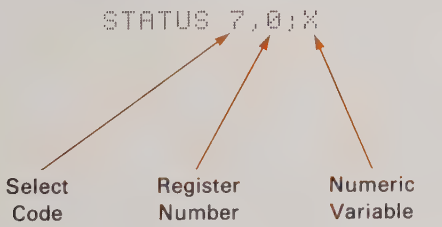
This section deals with the statements available to the programmer for tailoring the operations of an interface to the specific requirements of his system.

The status of interface operations can be monitored with the `STATUS` statement. This status may reflect the actual hi/low voltage level of external I/O lines or it may indicate the interface's internal state, depending upon which status register is being read.

The mode of operation of an interface can be directed by the `CONTROL` statement. The interface generally provides automatic control of external I/O lines acording to the mode selected, and also may provide for manual override of certain of the I/O lines for custom sequences.

Checking the Status

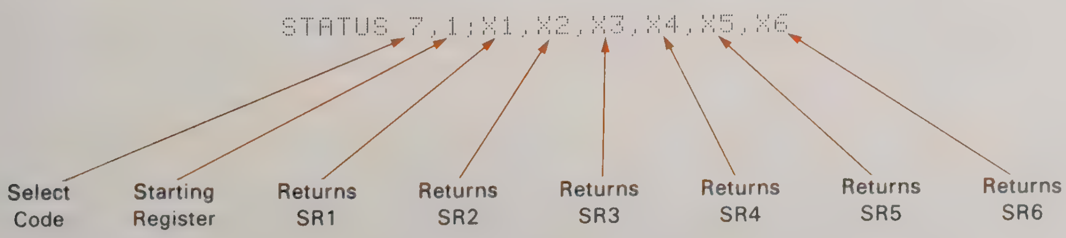
Interface status (and also buffer status) is monitored by the `STATUS` statement. The `STATUS` statement is a very straightforward one to use, requiring only the interface's select code and desired register number to be specified. The following statement obtains the value of the HP-IB interface (select code 7) identification register, SR0:



```
STATUS 7,0;X
```

In this case, the value returned is always 1, the identification code for an HP-IB interface. Each interface returns a different identification code, so consult the appropriate interface owner's manual for specific details.

The values of multiple status registers can be obtained with just one `STATUS` statement. Simply specify the first register number and give as many variables as necessary to obtain the status needed. For example, to read status registers SR1 through SR6 of the HP-IB interface at select code 7:



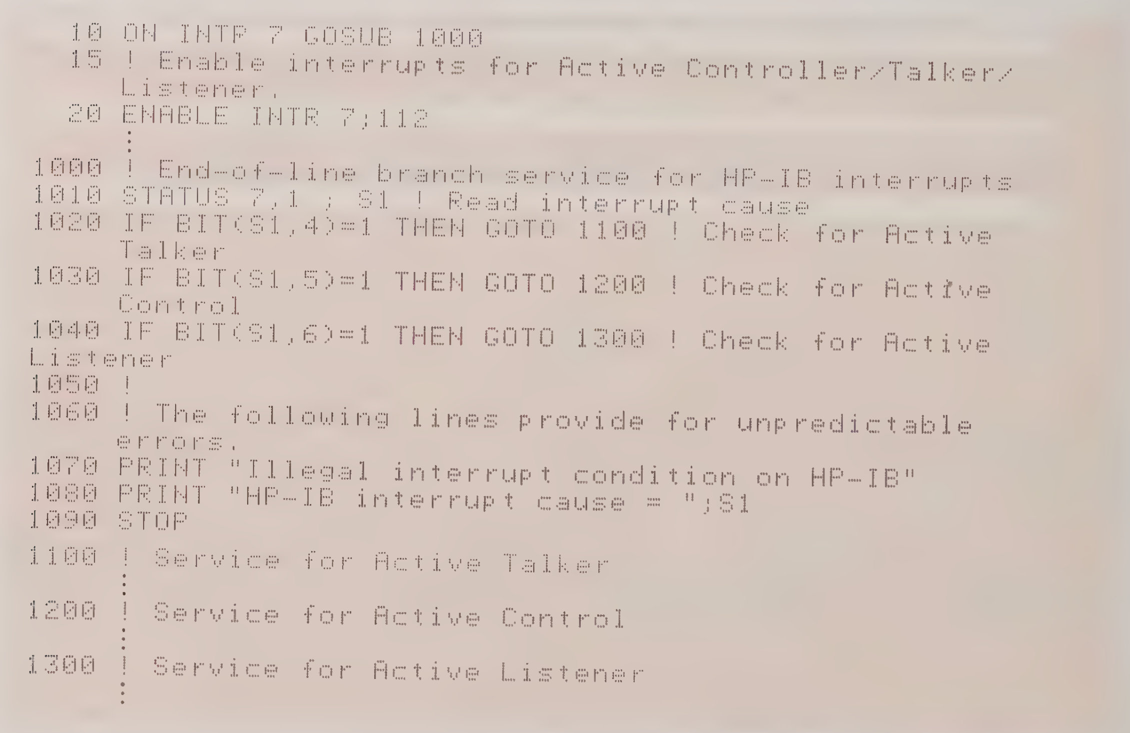
```
STATUS 7,1;X1,X2,X3,X4,X5,X6
```

What you do with the contents of these registers depends upon your application, needs, and type of interface. The HP-IB interface above, for example, returns

- Interrupt Cause—SR1
- HP-IB Control Lines—SR2
- HP-IB Data Lines—SR3
- HP-IB Address—SR4
- HP-IB State—SR5
- Secondary Command—SR6

for the six registers just read. Obviously the BCD Interface returns different information for those same registers.

Most interfaces are capable of interrupting the computer when a specified condition occurs, and if more than one condition has been enabled it will be necessary to determine which interrupt occurred and caused the end-of-line branch. The following service routine serves as an example of how `STATUS` would be so used:



```
  10 ON INTR 7 GOSUB 1000
  15 ! Enable interrupts for Active Controller/Talker/
     Listener.
  20 ENABLE INTR 7;112
     ⋮
1000 ! End-of-line branch service for HP-IB interrupts
1010 STATUS 7,1 ; S1 ! Read interrupt cause
1020 IF BIT(S1,4)=1 THEN GOTO 1100 ! Check for Active
     Talker
1030 IF BIT(S1,5)=1 THEN GOTO 1200 ! Check for Active
     Control
1040 IF BIT(S1,6)=1 THEN GOTO 1300 ! Check for Active
Listener
1050 !
1060 ! The following lines provide for unpredictable
     errors.
1070 PRINT "Illegal interrupt condition on HP-IB"
1080 PRINT "HP-IB interrupt cause = ";S1
1090 STOP
1100 ! Service for Active Talker
     ⋮
1200 ! Service for Active Control
     ⋮
1300 ! Service for Active Listener
     ⋮
```

Line 1010 obtains the interrupt cause from SR1 of the interface, and lines 1020-1040 analyze the register bits for the specific cause of the end-of-line branch. Lines 1050-1090 are included to deal with possible program errors or illegal interrupt causes. No program or machine is perfect, so try to make provisions such as the one shown to deal with malfunctions that might occur.

Interface Control

The operating modes of an interface can generally be tailored to suit individual systems and their requirements. This may mean nothing more than selecting the type of interrupt conditions required to deal with events specific to your system. Or, it may mean selecting a handshake mode suitable for the device being connected to the interface. In any case, the `CONTROL` statement provides the means of programming the interface's control registers.

There are three primary items of interest in a general discussion about `CONTROL` and interface control registers. The first is the interface interrupt mask. This mask is CR1 for the HP-IB Interface. Therefore, the following two statements have *identical effects* for the HP-IB Interface (select code 7):

```
ENABLE INTR 7;8
CONTROL 7,1;8
```

Both statements enable an SRQ interrupt condition for the interface. They can be used interchangeably, however, the `ENABLE INTR` statement better documents the effects it will have on program execution.

The second item of interest is that of external I/O lines. In general, you can set or clear interface-specific control lines by writing to control register CR2 of the desired interface (however, consult the individual interface manuals for exact details). Both of the following statements write to HP-IB control register CR2, the HP-IB Control Lines register (select code 7):

```
ASSERT 7;32
CONTROL 7,2;32
```

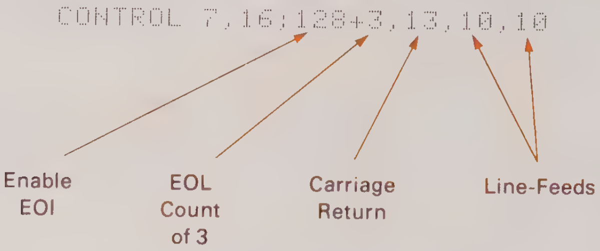
Both statements set the HP-IB SRQ (Service Request) control line true, but there is a slight difference in the manner that they do so. The `CONTROL` statement is not immediately executed if there is an I/O operation already in progress (an interrupt transfer, for example). The I/O in progress first completes, then the CONTROL operation is performed. This is in contrast to the `ASSERT` statement, which *immediately* sets the control line (SRQ in this case) regardless of any I/O in progress. In either case, exercise due caution when writing to the interface control register: the consequences of an improper control operation may be an interface or device malfunction. (In fact, to assert SRQ you should really use the `REQUEST` statement so that proper bus protocol is ensured.)

Note: Improper or invalid control line operation may cause loss of data or device malfunctions.

The third item of general interest in control register programming is the end-of-line (EOL) character sequence sent by the interface. This character sequence is essentially an end-of-record delimiter, which in the days of punched cards signaled that an entire card had been read or punched. For all interfaces, this EOL sequence defaults to carriage return/line feed, and is sent after every `PRINT` or `OUTPUT` operation (unless inhibited by the programmer). The EOL sequence is also sent for the `"/"` (slash) image specifier or for the `"END"` keyword of the `SEND` statement.

It is sometimes necessary to tailor the EOL sequence to suit the needs of a particular device. For instance, you might need double spacing on a printer (carriage return/double line feed) or carriage-return only for a CRT terminal that performs an automatic line-feed when a carriage-return is received.

The HP-IB Interface provides EOL sequence programmability with control registers CR16-CR23. Up to seven EOL characters can be sent if so desired, and are specified by writing the character values and EOL character count to those control registers. For example, to program a double line-feed EOL sequence with EOI (END or Identify) set on the last line-feed*, the following `CONTROL` statement could be used:



```
CONTROL 7,16;128+3,13,10,10
```

In summary, the control registers provide the flexibility required to tailor interface operation to your specific system requirements. They should be used with caution, however, so that you don't get any unpleasant surprises. Study the appropriate interface manual to determine which capabilities you need and how to implement them. It may be necessary to experiment a bit before the system works as you want it, but even the pros have to do that.

---

*You should note that it is not possible to send EOI with the last data byte of an `OUTPUT` or `SEND` operation, but it is possible with `PRINT`, `DISP`, or `TRANSFER`.

Additional I/O Commands

There are several commands made available by the I/O ROM which have not been introduced in the preceding sections. Many of these commands are only used with certain interfaces and all of them have different meanings with respect to each interface. Several of these interface-specific commands are introduced here to illustrate their capabilities. For extensive examples the interface owner's manuals should be consulted.

Interface-Dependent Statements

The `SEND` Statement

There are times when it is convenient to be able to send an arbitrary data sequence from the computer to one or more peripherals. The `SEND` statement accomplishes this directly. The following are examples of properly formed `SEND` data statements:

```
SEND 705; DATA "welcome to the world of I/O"
SEND 9; DATA "OCT. 9", "NOV. 13"
```

If EOL is specified, the interface sends an end-of-line sequence following each set in the data listing.

With all but the serial interface, the `SEND` statement may be used to send commands. This allows sending an arbitrary sequence of bytes out over the interface bus, which may be tailored to command any device which requires special sequences for initialization or reconfiguration. These same commands could be sent with `OUTPUT` statements, of course, but by using the `SEND` statement judiciously program documentation may be enhanced. The following are examples of correctly formed `SEND` command statements:

```
SEND 7; CMD B$
SEND 9; MTA MLA UNL LISTEN 4,5 CMD F,H
SEND 4; CMD X$
```

In most cases the interface and device owner's manuals will need to be consulted to confirm the character programming sequences needed.

`HALT`, `ABORTIO`, and `RESET`

The `HALT` statement is used to terminate any transfer in progress. This is the best means of recovering from a *hung* condition (interface handshake problem). This will not have any effect on a `FHS TRANSFER` and other means will be needed to recover from that type of condition.

The next most commonly used statement, `ABORTIO`, will also halt any transfers in progress and will reset control lines on all interfaces and will reset data lines on some interfaces. This is a convenient method of interrupting an I/O operation and ending up in a known state.

`RESET` is the most drastic statement of this group since it not only terminates any transfers but also returns all configurations selected to their default (switch-selected) state. This should only be necessary when first beginning a program or when recovering from an interface failure.

If an EOT branch is enabled when any of these statements is executed, the branch will be taken. After a `HALT` or `SEND` statement, a `RESUME` is sometimes necessary to re-enable input or output operations. The `CLEAR` command is used with the GPIO, HP-IB, and HP-IL interfaces to reset peripherals. The interface itself is not reset. Examples of these statements are given below:

```
HALT 7
ABORTIO 7
RESET 7
CLEAR 705
RESUME 7
```

Bus-Controlling Functions

With the instrumentation-controlling interfaces (HP-IB and HP-IL) the `REMOTE` and `LOCAL` statements can be used to allow or not allow access to device settings by use of the front-panel controls supplied on the device.

The `REMOTE` statement places instruments under remote control. The instrument may be returned to front panel control by executing a `LOCAL` statement or by activating a switch on the instrument called *Return to Local* or *Manual Control*.

If system security requires, a `LOCAL LOCKOUT` statement may be executed which precludes any local control whatsoever until a `LOCAL` statement is subsequently executed.

For an example of the `LOCAL` statement, see the program illustrating the Interface Interrupts section of section 6.

These statements will not be covered in this manual since they have specific meanings (quite specific!) with each interface they are to be used with:

```
REQUEST        SPOLL        PPOLL        PASS CONTROL        TRIGGER
```

Refer to the syntax reference in the first appendix of this manual for a brief summary of the uses of these statements with each interface. The interface owner's manuals will contain more specific instructions and examples.

Section 10

Binary Functions

Introduction

I/O programming often involves the use of binary functions and bit-level operations. The I/O ROM provides several useful tools to assist you with these tasks. These binary functions are often used when manipulating status and control registers in the interface cards. They may also be very handy for processing I/O data in device control applications. This section explains the binary functions available with the I/O ROM.

It is important to remember that all the binary functions provided by the I/O ROM operate on 16-bit words. For example, the binary complement of zero is 11111111 11111111 (base 2); the range of bits that can be tested is 0 thru 15; and the range of values for binary arguments is −32 768 thru 32 767. There is no problem using these functions to operate on binary values with less than 16 bits. The unused high-order bits are simply assumed to be zero.

In the following explanations, the term *integer* is used frequently to identify the arguments for many of the functions. To the I/O ROM, an integer is a 16-bit binary number with a range of −32,768 thru 32,767. This contrasts to the definition of an integer given in BASIC where an integer is a 5-digit number with a range of −99,999 thru 99,999. Please keep this distinction in mind to avoid confusion about the term "integer".

If you are not familiar with the binary numbers and operators, study the following review before continuing with this section.

Review of Base 2

Before looking at base 2, it is helpful to take a careful look at the familiar base 10. The number one hundred twenty five is represented as follows:

125

The digits have a *place value* corresponding to powers of ten. The representation above really means:

$$1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$$

The concept of place value also exists in base 2. The difference being that powers of two are represented instead of powers of ten. The number one hundred twenty five is represented as:

11111101

Base 2 uses only the digits "1" and "0"; a 1 indicates that a place value is included, while 0 indicates that a place value is not used in the value. Therefore, the binary representation shown above means:

$$2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0$$

This is the same as:

$$64 + 32 + 16 + 8 + 4 + 1$$

The term *bit* comes from the words "binary digit". A bit is a single digit in base 2 that must be either a 1 or a 0. The grouping of 8 bits together is in such common usage for character representation, internal storage, and interfacing that it has been given a special name—a **byte**. The term byte refers to 8 bits processed as a unit.

Notice in both the previous examples that the right-most digit represents the "0th" power of the base. Because of this, bit patterns are usually numbered starting at Bit 0, instead of Bit 1. By doing this, the bit number and the power of two it represents are the same. The following table shows the bit positions in a byte and their corresponding values.

Bit Position	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Meaning	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value	128	64	32	16	8	4	2	1

Examples: 130 in base 10 is 10000010 in base 2
3 in base 10 is 00000011 in base 2
25 in base 10 is 00011001 in base 2

The term *word* is also widely used in computer appliations. A "word" is usually the number of bits that can be handled in one operation by the internal architecture of the computer. Although the Series 80 Personal Computers have an 8-bit internal architecture, they also have operations defined for 16-bit and floating point numbers. Therefore, 16-bit integers are often referred to as words in the computer because the system can handle them as a basic data unit. Another characteristic of a word in the computer is that *2's complement* representation is used. 2's complement representation is a method of storing either positive or negative numbers in a word. It works like this:

Positive numbers: If bit 15 is 0, then the word is a positive number represented in normal binary form.

Negative numbers: If bit 15 is 1, then the word is a negative number represented in 2's complement form. To find the absolute value of a negative number, invert all the bits and add 1.

Problem: What is the value of 11111111 11110000?

Solution: Bit 15 tells that this is a negative number.
Inverting all the bits gives: 00000000 00001111
Adding 1 results in: 00000000 00010000
So the value of the given bit pattern is −16.

Review of Logical Operations

In this discussion, "logical operations" refers to operations from Boolean algebra, such as AND and OR. The outstanding I/O feature of these operations is that they can modify individual bits without affecting surrounding bits. In this respect, they can be contrasted to the arithmetic operations, such as addition and subtraction. Addition and subtraction generate carries and borrows that can propagate through an entire word, changing the state of numerous bits many places away from the bit where the arithmetic was performed. Although this is exactly what is desired for numerical quantities, many of the bytes and words used in I/O are not numerical quantities. When bits are used as individual control elements, the programmer must have access to tools that allow individual control of bits. This section reviews the action of the common logical operators.

The operators AND, OR, NOT, and EXOR are available in the standard language of the HP Series 80 computers. These operators treat an entire variable as one entity. A value of "0" is considered "false", while any other value is considered "true". Although these operators perform the same Boolean function as the binary logical operators, they do not operate on individual bits. The binary logical operators available with the I/O ROM work on a bit-by-bit basis across an entire word. Without these binary tools, isolating an individual bit requires an involved combination of tests, branches, and arithmetic operators.

The simplest logical operation is the *complement* operation. When binary data is complemented, all the 1's are changed to 0's, and all the 0's are changed to 1's. This operation is also known as "1's complement", or "inversion". The Boolean notation for this operation is a horizontal bar drawn over the variable. The truth table is as follows:

A	$\overline{A}$
0	1
1	0

When used on an entire byte, the complement operator inverts each bit individually.

Binary value of A:	10011101
Binary complement of A:	01100010

The other logical operators combine two inputs to create a result. Let's look at the AND operator first. A binary AND produces a "1" in the result only if both inputs are "1". The Boolean notation for this operation is ^, although you may also see the symbol · used. The truth table for AND is as follows:

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

The important thing to notice is that the result is 0 when A is 0, while the result is equal to B when A is 1. Because of this, a binary AND is a convenient method for clearing selected bits. For example, assume that you wanted to clear the two lowest bits in a byte without disturbing the other bits. This can be done by ANDing the byte with an appropriate mask.

Original bute:	10011101
Byte ANDed:	11111100
Result:	10011100

This operation not only preserves the state of the top six bits, but also clears the bottom two bits no matter what their original state. That saves alot of testing and branching.

The next operator is the binary OR, most correctly called the *inclusive OR*. In English this means: you can have pie OR ice cream for dessert, and it is possible to have both at the same time. To the computer this means that the result bit is "1" when either input bit is "1". The Boolean notation for an inclusive OR is v, although you may also see the symbol + used. The inclusive OR truth table is:

A	B	A v B
0	0	0
0	1	1
1	0	1
1	1	1

The important thing to notice is that the result is 1 when A is 1, while the result equals B when A is 0. Because of this, the inclusive OR is a convenient method for setting selected bits. For example, assume you wanted to set the two lowest bits in a byte without disturbing the other bits. This can be done by ORing the byte with an appropriate mask.

Original byte:	10011101
Byte ORed:	00000011
Result:	10011111

This operation sets the lower two bits no matter what their original state and also preserves the state of the top six bits.

The final operator is the binary EXOR, or *exclusive OR*. In English this means: you can take the plane OR the train to Chicago, but you can't do both at the same time. To the computer this means that the result bit is "1" if a single input bit is "1", but the result bit is "0" if both input bits are the same. The Boolean notation for an exclusive OR is v, although you may see the symbol + used. The exclusive OR truth table is:

A	B	A v B
0	0	0
0	1	1
1	0	1
1	1	0

This important thing to notice is that the result is equal to B when A is 0, while the result is the complement of B when A is 1. Because of this, the exclusive OR is a convenient method for inverting selected bits. For example, assume that you wanted to invert the lower two bits of a byte without disturbing the rest of the bits.

Original byte:	10011110
Byte EXORed:	00000011
Result:	10011101

This operation complements the lower two bits no matter what their original value and leaves the top six bits unchanged.

The Binary AND Function

The binary AND function performs a bit-by-bit AND using two integers as arguments and producing an integer result. Here are some examples of properly formed binary AND functions:

```
X=BINAND(Y,15)
A=C+BINAND(D,E)
PRINT BINAND(255,Z-32)
```

Notice that the arguments must be enclosed in parentheses and separated by a comma. The arguments may be numeric constants, numeric variables, numeric expressions, or any combination. The arguments are assumed to be in base 10 representation. If you wish to express the arguments in another base, refer to section 11. Each bit of the result is computed according to the following truth table:

First Argument	Second Argument	Function Result
0	0	0
0	1	0
1	0	0
1	1	1

The Binary Inclusive OR Function

The binary inclusive OR function performs a bit-by-bit inclusive OR using two integers as arguments and producing an integer result. Here are some examples of properly written binary inclusive OR functions:

```
X=BINIOR(Y,15)
A=C+BINIOR(D,E)
PRINT BINIOR(255,Z-32)
```

Notice that the arguments must be enclosed in parentheses and separated by a comma. The arguments may be numeric constants, numeric variables, numeric expressions, or any combination. The arguments are assumed to be in base 10 representation. If you wish to express the arguments in another base, refer to section 11. Each bit of the result is computed according to the following truth table:

First Argument	Second Argument	Function Result
0	0	0
0	1	1
1	0	1
1	1	1

The Binary Exclusive OR Function

The binary exclusive OR function performs a bit-by-bit exclusive OR using two integers as arguments and producing an integer result. Here are some examples of correctly stated binary exclusive OR functions:

```
X=BINEOR(Y,15)
A=C+BINEOR(D,E)
PRINT BINEOR(255,Z-32)
```

Notice that the arguments must be enclosed in parentheses and separated by a comma. The arguments may be numeric constants, numeric variables, numeric expressions, or any combination. The arguments are assumed to be in base 10 representation. If you wish to express the arguments in another base, refer to section 11. Each bit of the result is computed according to the following truth table:

First Argument	Second Argument	Function Result
0	0	0
0	1	1
1	0	1
1	1	0

The Binary Complement Function

The binary complement funcion performs a bit-by-bit complement of an integer argument, producing an integer result. Here are some examples of properly formed binary complement functions:

```
A=BINCMP(B)
X=BINCMP(Y-Z)
PRINT BINCMP(N*8)
```

Notice that the argument is enclosed in parentheses. The argument may be a numeric constant, a numeric variable, a numeric expression, or any combination. The argument is assumed to be in base 10 representation. Each bit of the result is computed according to the following truth table:

Argument	Result
0	1
1	0

You should keep in mind that the binary complement function operates on a full 16-bit word. This may, in some cases, give an unexpected result if you are dealing exclusively with 8-bit bytes. The 16-bit complement of an 8-bit byte is always a negative number. You can generate 8-bit complements by using the binary exclusive OR function. An exclusive OR with 255 complements the lower eight bits and leaves the upper eight bits as zeros. This technique prevents the unintentional generation of negative values when dealing with single bytes.

The Bit Test Function

The bit test function is used to indicate whether a specific bit in an integer is set (1) or clear (0). The general form for the bit test instruction is:

```
BIT(integer, bit number)
```

The integer argument must be the first expression and the two expressions must be separated by a comma. The bit number must be in the range 0 thru 15, where 0 is the least-significant bit and 15 is the most-significant bit. Either argument may be a numeric constant, a numeric variable, a numeric expression, or any combination. Here are some examples of properly stated bit test functions:

```
A=BIT(B,3)
X=BIT(Y,Z-1)
IF BIT(N,1) THEN GOSUB 220
```

The bit test function is very useful in decision making and branching. It is easily used with the `IF` statement to direct program flow based on the state of individual bits. The function returns a 0 (false) if the specified bit is 0 and returns 1 (true) if the specified bit is 1.

Section 11

Base Conversion Functions

Introduction

A programmer who works at the bit and byte level soon develops a preference for the base in which bytes and words are represented. In some cases, base 2 offers the clearest display of a bit pattern. Base 8 had a large following in the days when computers could not easily handle alphabetic characters as numeric input. Base 16 has gained much popularity in recent years because most computers use a word length that is an integral multiple of 4, and modern systems have no trouble converting the symbols A thru F used in base 16, also known as *hexadecimal* or simply *hex*.

To accommodate these various preferences, your computer provides conversion functions that allow theinput and output of integers using any of the alternate representations mentioned above. The base conversion functions have certain characteristics in common:

- All conversions go from base 10 to an alternate base or from an alternate base to base 10. You can't convert directly from one alternate base to another without passing through base 10.
- The base 10 side of the conversion is always a numeric quantity, while the alternate base side is always a string.
- Because the alternate base representations are string data, they can be input, output, compared, stored, and manipulated to some degree. However, the string representations cannot be used in arithmetic operations.
- All arguments for the base conversion functions must be in the range of 16-bit integers. This includes the alternate representations as well as the base 10 values.

If you are not familiar with alternate number bases, read the following review material.

Review of Alternate Representations

When text contains values represented in more than one base, it is extremely important to distinguish between the concepts of *value* and *representation*. Consider the number one hundred. The value is the number of beans in a jar of one hundred beans. The representation in base 10 is the digit "1" followed by two zeros. The value one hundred can also be represented as "64" in base 16, as "01100100" in base 2, and as "10" in base 100.

The representation of a number is merely the character set used to communicate the number's value. Numbers are often represented in bases other than 10 when the use of an alternate base more clearly communicates the number's value. For example, suppose that up to 16 small pumps and valves are controlled by a single 16-bit word from the computer. If the control pattern were represented in base 10, it could be very difficult to understand the number in terms of pumps and valves. However, suppose the number is represented in base 2, further defined so that the most-significant byte is pumps and the least-significant byte is valves. The base 2 representation "00100000 10000000" clearly shows one pump and one valve open. That same value is "8320" in base 10. How quickly does "8320" communicate to you that one pump is on and one valve is open?

The problem with using base 10 to represent a binary number is that one base 10 digit does not represent an integral number of bits. A base 10 pattern does not readily reflect which bits are "1" and which are "0". The problem with using base 2 is that there are simply too many characters to read and write. To circumvent these problems, persons who work at the bit and byte level in computers commonly use base 8 or base 16 to represent binary numbers. These bases have place values directly related to powers of two, making it easy to trace bits with a little practice. They also provide representations that are three and four times more compact than binary, reducing the number of characters needed to a more manageable small group. For example, an entire byte is never more than 2 characters in base 16.

Base 8, known as *octal*, uses one octal digit for three binary digits. Base 16, known as *hex*, uses one hex digit for four binary digits. The following tables show the decimal (base 10), binary (base 2), octal (base 8), and hex (base 16) representations for the numbers 0 thru 16.

Decimal	Binary	Octal	Hex
0	000	0	0
1	001	1	1
2	010	2	2
3	011	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1 000	10	8
9	1 001	11	9
10	1 010	12	A
11	1 011	13	B
12	1 100	14	C
13	1 101	15	D
14	1 110	16	E
15	1 111	17	F
16	10 000	20	10

Notice that the Arabic numeral system (designed for base 10) does not have any single-character symbols to represent quantities above nine. Single-character representation of all quantities less than the base value is essential to the concept of place value. Therefore, base 16 representation utilizes the characters A thru F to represent values from 10 thru 16. The following examples help to illustrate the way that octal and hex numbers use bit groupings to represent binary values.

Octal:	1 2 5	3 6 0	0 7 0	2 0 1	2 0 2
Binary:	01010101	11110000	00111000	10000001	10000010
Hex:	5 5	F 0	3 8	8 1	8 2

Conversions From Base 10 to an Alternate Base

These functions use a base 10 numeric quantity as an argument and produce a string as a result. The primary use of these functions is the printing, display, or output data in an alternate base, although other applications are possible. The argument for the function may be a numeric constant, numeric variable, numeric expression, or any combination. The argument must be in the range of −32 768 thru 32 767. Functions are available to convert to base 2, base 8, or base 16.

From Base 10 to Base 2

This is the **Decimal To Binary String** function. It converts an integer argument to a string of 16 ones and zeros. The string is the base 2 representation of the integer argument. If the argument is out of range and positive, the function yields "0111111111111111". If the argument is out of range and negative, the function yields "1000000000000000". The following are examples of well-stated conversion expressions:

```
PRINT DTB$(X)
A$=DTB$(N*2)
OUTPUT 701;DTB$(32+Y)
DISP DTB$(255)
```

From Base 10 to Base 8

This is the **Decimal To Octal String** function. It converts an integer argument to a 6-character string. The string is the octal representation of the integer argument. If the argument is out of range and positive, the function yields "077777". If the argument is out of range and negative, the function yields "100000". The following are examples of properly formed decimal-to-octal expressions:

```
PRINT DTO$(X)
A$=DTO$(N*2)
OUTPUT 701;DTO$(32-Y)
DISP DTO$(255)
```

From Base 10 to Base 16

This is the **Decimal To Hex String** function. It converts an integer argument to a 4-character string. The string is the hex (hexadecimal) representation of the integer argument. If the argument is out of range and positive, the function yields "7FFF". If the function is out of range and negative, the function yields "8000". The following are examples of correct decimal-to-hex expressions:

```
PRINT DTH$(X)
A$=DTH$(N*2)
OUTPUT 701;DTH$(32-Y)
DISP DTH$(255)
```

Conversions From an Alternate Base to Base 10

These functions use a string as an argument and produce a numeric result. The primary use of these functions is the input of data in an alternate base, although other applications are possible. The argument for the function may be a string constant (literal), string variable, string expression, or any combination. The argument must represent a value in the range of 16-bit integers. Functions are available to convert from base 2, base 8, or base 16.

From Base 2 to Base 10

This is the **Binary To Decimal** function. The argument is a string which is the binary representation of an integer. The argument cannot have more than 16 characters, and only the numerals "1" and "0" are valid. The result of the function is the base 10 value of the number represented by the argument. Since the function result is numeric, it can be used in arithmetic operations or numeric functions. The following are examples of properly formed expressions:

```
PRINT BTD("100101")
X=BTD(A$)
Y=255-BTD(N$)
```

From Base 8 to Base 10

This is the **Octal To Decimal** function. The argument is a string which is the octal representation of an integer. The argument cannot have more than 6 characters. Only the numerals "0" thru "7" are valid. If all six characters are used, the most-significant character can only be a "1" or a "0". The result of the function is the base 10 value of the number represented by the argument. Since the function result is numeric, it can be used in arithmetic operations or numeric functions. The following are examples of correctly stated expressions:

```
PRINT OTD("371")
X=OTD(A$)
Y=255-OTD(N$)
```

From Base 16 to Base 10

This is the **Hex To Decimal** function. The argument is a string which is the hex representation of an integer. The argument cannot have more than four characters. Only the numerals "0" thru "9" and the letters "A" thru "F" are valid. The result of the function is the base 10 value of the number represented by the argument. Since the function result is numeric, it can be used in arithmetic operations or numeric functions. The following are examples of properly written hex-to-decimal expressions:

```
PRINT HTD("1F4")
X=HTD(A$)
Y=255-HTD(N$)
```

Converting From One Alternate Base to Another

Conversions between alternate representations are easily done by nesting two conversion functions. The following short program is an example of this technique. It inputs a hex representation and displays the corresponding binary representation.

```
10 DISP "Hex Number";
20 INPUT A$
30 DISP "Binary = ";DTB$(HTD(A$))
40 GOTO 10
```

Appendix A

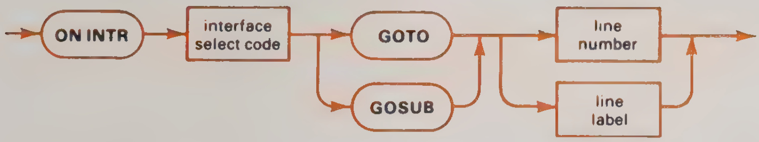
Syntax Reference

Conventions Used to Represent Syntax

This reference section uses two methods of representing the syntax of I/O ROM statements. The conventions of each form are as follows.

Pictorial Representation

All items enclosed by a rounded envelope must be entered exactly as shown. Items enclosed by a rectangular box are names of parameters used in the statement. A description of each parameter is given in the text following the drawing. Statement elements are connected by lines. Each line can only be followed in one direction, as indicated by the arrow at the end of the line. Any combination of statement elements that can be generated by following the lines in the proper direction is syntactically correct. A statement element is optional if there is a valid path around it.



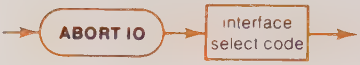
This form of syntax representation is easy to use, and in some cases, more formally correct than the alternate form described next.

Linear Representation

This form of syntax representation is included to be compatible with previous HP Series 80 manuals. Many users are accustomed to seeing this form.

`DOT MATRIX`	All items shown in dot matrix must be entered exactly as shown.
[ ]	Items within square brackets are optional.
item 1 *item 2*	A vertical placement of two items indicates that only one of the items may be included.
...	Three dots indicate that successive parameters are allowed.
italic	Italicized items are the parameters themselves.

ABORTIO



`ABORTIO` *interface select code*

Example Statements

```
100 ABORTIO 7
250 IF S<128 THEN ABORTIO S0
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

Actions Taken

All interfaces: Terminates any interrupt transfer in progress. Performing an `ABORTIO` on an interface with an active transfer and EOT branching enabled causes the branch to be taken.

HP-IB:

- System Controller: Sends Interface Clear (IFC) and Remote Enable (REN).
- Active Controller (but not System Controller): Sends Attention (ATN) and My Talk Address (MTA).
- Non controller: Stops handshaking data and becomes ready for next operation.

Serial: Turns off all modem control lines (control register 2).

BCD: Stops handshaking data, sets CTL line false, and places external data lines in high-impedance state.

GPIO: Stops handshaking data, sets control lines false, places ports A and B in high-impedance state, and sets lines from ports C and D to false state.

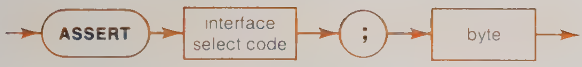
HP-IL:

- System or Active Controller: Sends Interface Clear.
- Non controller: Stops current operation and becomes ready for next operation.

Related Statements

```
HALT
ON EOT
RESET
```

ASSERT



`ASSERT` *interface select code* `;` *byte*

Example Statements

```
100 ASSERT 7 ; 12
210 IF A1=128 THEN ASSERT 8 ; X
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

byte—a numeric expression that evaluates to an integer 0 thru 255. Binary value of the byte is used to set or clear the lines to be asserted.

Actions Taken

HP-IB: Immediately writes the value of the byte to control register 2. IFC bit (2^7, decimal 128) is ignored (use `ABORTIO`).

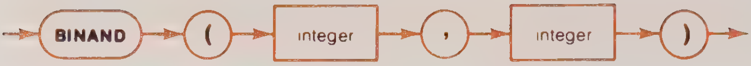
Serial, BCD, GPIO: Immediately writes the value of the byte to control register 2.

HP-IL: Sends a frame using the specified byte and the most recent control bits written to register 2. This statement is similar to `CONTROL` *isc* `, 3 ;` *byte* except that `ASSERT` interrupts the interface and sends the frame without checking loop status.

Related Statements

```
ABORTIO
CONTROL
```

BINAND



`BINAND` *(integer, integer)*

Example Statements

```
10 B1=BINAND(X1,15)
100 PRINT BINAND(I,N*2^3)
```

Parameters

integer—a numeric expression that evaluates to an integer −32,768 thru 32,767.

Action Taken

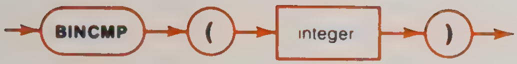
`BINAND` is a function that returns the 16-bit binary AND of two integer values. Each bit of the result is calculated using the corresponding bit of each argument, according to the following truth table:

Arg. 1	Arg. 2	Result
0	0	0
0	1	0
1	0	0
1	1	1

Related Statements

```
BINCMP
BINEOR
BINIOR
BIT
```

BINCMP



BINCMP (*integer*)

Example Statements

```
100 C=BINCMP(X1)
120 PRINT BINCMP(N*2^3)
```

Parameters

integer—a numeric expression that evaluates to an integer −32,768 thru 32,767.

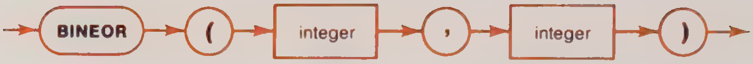
Action Taken

`BINCMP` is a function that returns the 16-bit binary complement of an integer value. Each bit of the result is the inverse of the corresponding bit in the argument. If the argument has less than 16 bits, leading zeros are assumed.

Related Statements

```
BINAND
BINEOR
BINIOR
BIT
```

BINEOR



`BINEOR` (*integer*, *integer*)

Example Statements

```
20 B1=BINEOR(X1,15)
140 PRINT BINEOR(I,2^N)
```

Parameters

integer—a numeric expression that evaluates to an integer −32,768 thru 32,767.

Action Taken

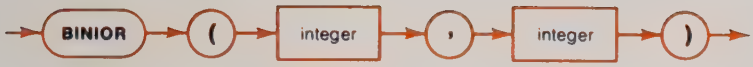
`BINEOR` is a function that returns the 16-bit binary exclusive OR of two integer values. Each bit of the result is calculated using the corresponding bit of each argument, according to the following truth table:

Arg. 1	Arg. 2	Result
0	0	0
0	1	1
1	0	1
1	1	0

Related Statements

```
BINAND
BINCMP
BINIOR
BIT
```

BINIOR



`BINIOR (`*integer* `,` *integer*`)`

Example Statements

```
30 Y=BINIOR(X1,255)
160 DISP BINIOR(I,2^N)
```

Parameters

integer—a numeric expression that evaluates to an integer −32,768 thru 32,767.

Action Taken

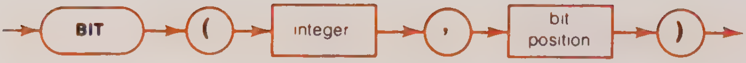
`BINIOR` is a function that returns the 16-bit binary inclusive OR of two integer values. Each bit of the result is calculated using the corresponding bit of each argument, according to the following truth table:

Arg. 1	Arg. 2	Result
0	0	0
0	1	1
1	0	1
1	1	1

Related Statements

```
BINAND
BINCMP
BINEOR
BIT
```

BIT



BIT (*integer*, *bit position*)

Example Statements

```
40 Y=BIT(X3,7)
180 IF BIT(N,2^I) THEN GOTO 220
```

Parameters

integer—a numeric expression that evaluates to an integer −32,768 thru 32,767.

bit position—a numeric expression that evaluates to an integer 0 thru 15. Least-significant bit is in position 0, most-significant in position 15.

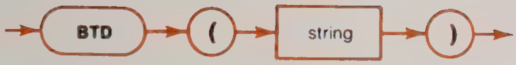
Action Taken

BIT is a function that returns the value of one bit in an integer argument. Result of the function is TRUE if bit is set, FALSE if bi is clear.

Related Statements

BINAND
BINCMP
BINEOR
BINIOR

BTD



BTD (*string*)

Example Statements

```
20 X=BTD(H$&L$)+A1
130 DISP BTD("11000001")
```

Parameters

string—a string expression that contains the base 2 representation of an integer. Limited to 16 significant characters that must be "1" or "0".

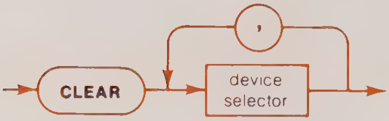
Action Taken

BTD is a function that returns the value of a base 2 representation contained in the string argument. The argument is a character representation and the result is a numeric quantity.

Related Statements

DTB$
DTH$
DTO$
HTD
OTD

CLEAR



`CLEAR` *device selector* [, *device selector*] ...

Example Statements

```
60 CLEAR 3
250 CLEAR S*100+D1,S*100+D2
```

Parameters

device selector—a valid interface select code or a valid combination of interface select code and primary address (see Choosing the Source or Destination in section 1). If multiple device selectors are specified, they must all be on the same interface select code.

Actions Taken

HP-IB and HP-IL: Must be Active Controller. (HP-IB leaves ATN true; use `RESUME` if you wish to set ATN false.)

- If device selector is only an interface select code, then Device Clear (DCL) is sent.
- If device selector contains a primary address, then Unlisten (UNL), Listen Address(es) (LAD), and Selected Device Clear (SDC) are sent.

Serial, BCD: Error

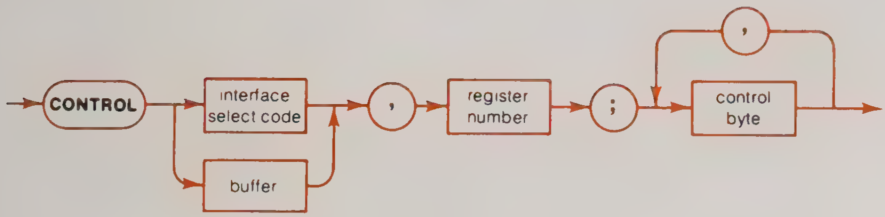
GPIO:

- If device selector is only an interface select code, interface pulses RESA and RESB.
- If device selector contains an even primary address, interface pulses RESA.
- If device selector contains an odd primary address, interface pulses RESB.

Related Statements

```
CONTROL
SEND
```

CONTROL



`CONTROL` *buffer*
interface select code , *register number* ; *control byte* [, *control byte*] ...

Example Statements

```
10 CONTROL 9,1 ; C1,C2,C3,C4,C5,C6
50 CONTROL S,R ; C(R)
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

buffer—the name of a string variable that has been declared an `IOBUFFER`.

register number—a numeric expression that evaluates to an integer 0 thru 23. Must specify a valid control register for the selected interface.

control byte—a numeric expression that evaluates to an integer 0 thru 255. Binary value of byte is used to set and clear bits in the control register.

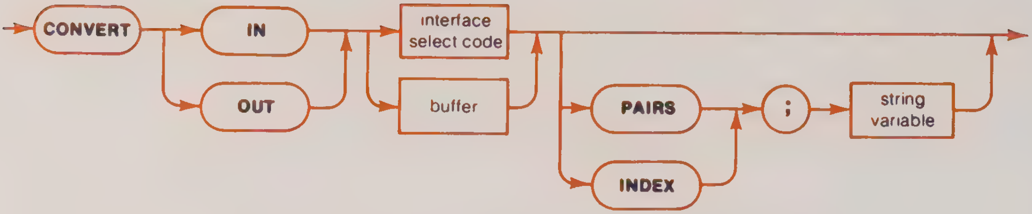
Action Taken

`CONTROL` writes one or more control bytes to interface or buffer control registers. The register number specifies the first register to be used. If multiple control bytes are specified, they are stored in consecutive control registers, beginning with the specified register number.

Related Statements

```
ABORTIO
ASSERT
ENABLE INTR
IOBUFFER
STATUS
```

CONVERT



CONVERT IN *buffer* [PAIRS
OUT *interface select code* INDEX ; *string variable*]

Example Statements

```
20 CONVERT OUT 4 PAIRS ; A$
50 CONVERT IN 10 INDEX ; C$
110 CONVERT IN 7 ! Turn off conversion
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

buffer—the name of a string variable that has been declared an IOBUFFER.

string variable—the name of a string variable in which the conversion table has been previously stored.

Actions Taken

Enables or disables a character conversion process for a specified interface or buffer and a specified direction. Although you can CONVERT using either an interface or a buffer, conversions can only be performed with OUTPUT and ENTER statements. Conversions are **not** performed during SEND or TRANSFER.

If the optional parameters are not included (as in the 3rd example statement), a previously selected conversion is turned off for the specified interface and direction.

If direction is specified as IN, all bytes being input from the specified source are processed through a conversion table immediately after they are received from the source. If direction is specified as OUT, all bytes being output to the specified destination are processed through a conversion table immediately before they are sent to the destination. IN and OUT conversions may both be specified for a given interface select code or buffer.

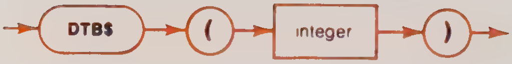
If conversion method PAIRS is specified, the conversion table is treated as a sequential list of character pairs, the second character in each pair being substituted for the first character. If the byte to be converted is not found as one of the first characters in a pair, it is passed through unchanged. Recommended when only a few characters need to be converted.

If the conversion method `INDEX` is specified, the numeric value of the byte to be converted is used as an index into the conversion table. The byte found as a result of this indexed lookup is substituted for the original byte. If the index value is greater than the length of the table, no conversion is performed. The first character in the string corresponds to the index value of 0. Recommended when a large number of characters need to be converted.

Related Statements

```
ENTER
IOBUFFER
OUTPUT
```

DTB$



DTB$(*integer*)

Example Statements

```
100 A$=DTB$(16+2*N)
200 PRINT DTB$(X1)
```

Parameters

integer—a numeric expression that evaluates to an integer −32,768 thru 32,767.

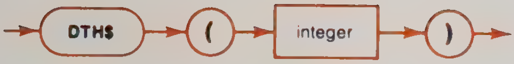
Action Taken

`DTB$` is a function that returns the base 2 representation of an integer argument. The result is a 16-character string and the argument is a numeric quantity.

Related Statements

```
BTD
DTH$
DTO$
HTD
OTD
```

DTH$



`DTH$(`*integer*`)`

Example Statements

```
110 B$=DTH$(32+2^N)
210 PRINT DTH$(X2)
```

Parameters

integer—a numeric expression that evaluates to an integer −32,768 thru 32,767.

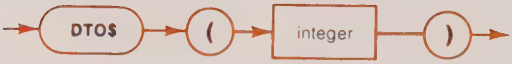
Action Taken

`DTH$` is a function that returns the base 16 representation of an integer argument. The result is a 4-character string and the argument is a numeric quantity.

Related Statements

```
BTD
DTB$
DTO$
HTD
OTD
```

DTO$



`DTO$(`*integer*`)`

Example Statements

```
120 C$=DTO$(64+2^N)
220 PRINT DTO$(X3)
```

Parameters

integer—a numeric expression that evaluates to an integer −32,768 thru 32,767.

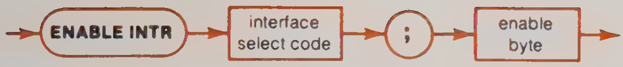
Action Taken

`DTO$` is a function that returns the base 8 representation of an integer argument. The result is a 6-character string and the argument is a numeric quantity.

Related Statements

```
BTD
HTD
OTD
DTB$
DTH$
```

ENABLE INTR



`ENABLE INTR` *interface select code* `;` *enable byte*

Example Statements

```
10 ENABLE INTR 7 ; 8 ! SRQ interrupt, HP-IB
50 IF S>2 AND S<11 THEN ENABLE INTR S ; X
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

enable byte—a numeric expression that evaluates to an integer 0 thru 255. Binary value of bytes is used to set and clear bits in the control register.

Action Taken

Enables the specified interface for interrupts according to the bits set in the enable byte. The enable byte is placed in control register CR1. The meaning of each bit in CR1 is interface dependent; refer to the appropriate interface owner's manual for details. This statement is identical to performing a `CONTROL` statement to control register 1.

Related Statements

```
CONTROL
ON INTR
STATUS
```

ENABLE KBD

ENABLE KBD
mask

```
ENABLE KBD mask
```

Example Statements

```
30 ENABLE KBD 33
180 IF X THEN ENABLE KBD K1
```

Parameters

mask—a numeric expression that evaluates to an integer 0 thru 255. Binary value of byte determines which keyboard modes are enabled and disabled.

Action Taken

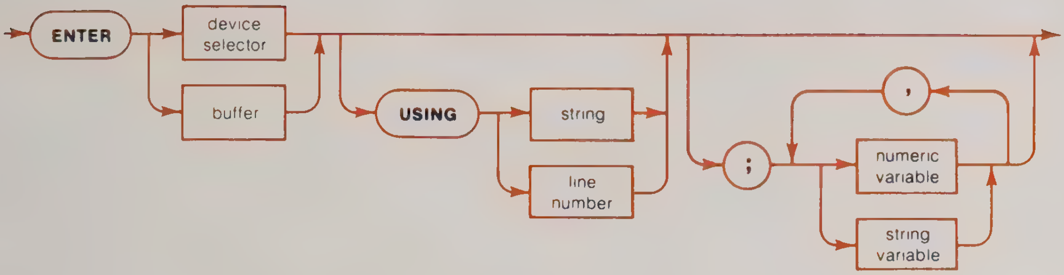
Bits in the mask byte correspond to various keyboard areas and program modes as shown in the following table. If a bit is set in the mask, its feature is enabled. If a bit is clear, its feature is disabled.

Bit	Mode	Keys Affected
7	RUN	RESET
6	RUN	PAUSE
5	RUN	SFKs and KEY LABEL
4	RUN	All other keys
3	INPUT	RESET
2	INPUT	PAUSE
1	INPUT	SFKs and KEY LABEL
0	INPUT	All other keys

Related Statements

```
INPUT
ON KEY#
```

ENTER



```
                                     string
ENTER       buffer         [USING line number ][; [variable][, variable]...]
       device selector              line label*
```

Example Statements

```
70 ENTER 701 USING LBL: ; X,Y,Z
90 ENTER C$ ; N(I),Z$
120 ENTER 3 USING 30 ; A$
250 ENTER 100*S+A USING "#,B" ; N
```

Parameters

device selector—a valid interface select code or a valid combination of interface select code and primary address (see Choosing the Source or Destination in section 1).

buffer—the name of a string variable that has been declared as an `IOBUFFER`.

string—a string expression that contains a valid set of image specifiers.

line number/label—the line number or label or an `IMAGE` statement that contains a valid set of image specifiers.

variable (numeric or string)—the name of a variable intended as a destination of the `ENTER` operation.

Action Taken

Inputs bytes from the specified buffer or device; uses those bytes to build a number or string; places the result into a BASIC variable. If a `CONVERT` is in effect, the conversion occurs immediately after the character is taken from the interface or buffer.

---

*Line labels are allowed on the HP-87 but not on the HP-85 or 83.

When USING is not specified, free-field format is used. A free-field entry into a string places incoming bytes into the variable until either a line feed is received, a carriage return/line feed sequence is received, or the string is full. Terminating sequences are not placed into the destination string. A free-field entry into a numeric variable ignores up to 256 leading non-numeric characters. Blanks are ignored during number building. Entry into a numeric variable is terminated by the first trailing character that is non-blank and non-numeric.

When USING is specified, input operations are formatted according to the image specifiers used. Image specifiers may be enclosed in quotes and placed in the ENTER statement, contained in a string variable named in the ENTER statement, or placed in an IMAGE statement referenced by the ENTER statement. For detailed information on image specifiers, refer to the IMAGE statement in this appendix or see Formatted ENTER in section 3.

ENTER requires a line-feed character to satisfy the statement after the variable list has been satisfied. This can be the same line-feed that satisfied the last variable in the list. If the source is a device selector and no line feed is detected, the computer will be "hung" on the ENTER statement. If the source is a buffer and no line feed is detected, a NO TERM error is generated. This requirement can be removed by using "#" as the first image specifier. For more detailed information on statement terminators, see Formatted ENTER. A "hung" condition can be trapped by use of the SET TIMEOUT and ON TIMEOUT statements.

Related Statements

CONVERT
IMAGE
IOBUFFER
ON TIMEOUT
SET TIMEOUT
TRANSFER

ERROM

```
ERROM
```

Example Statements

```
30 X=ERROM
70 IF ERROM=192 THEN GOTO 100
```

Parameters

None

Action Taken

`ERROM` is a function that returns the ID number of the option ROM associated with the last error generated by an option ROM. All option ROMs use error numbers greater than 100. The ID number of the I/O ROM is 192. Note that `ERROM` is modified only by the occurrence of another option ROM error.

Related Statements

```
ERRL
ERRN
ERRSC
```

ERRSC

```
ERRSC
```

Example Statements

```
40 Y=ERRSC
90 IF ERRSC=7 THEN GOSUB 200
```

Parameters

None

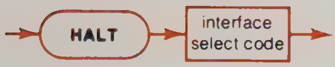
Action Taken

`ERRSC` is a function that returns the interface select code responsible for the most recent I/O error. Note that `ERRSC` is not cleared by a system reset and is modified only by the occurrence of another interface-dependent I/O error.

Related Statements

```
ERRL
ERRN
ERROM
```

HALT



`HALT` *interface select code*

Example Statements

```
100 HALT 7
200 HALT S1
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

Actions Taken

All interfaces: Stops current I/O operation. If an interface is `HALT`ed with a `TRANSFER` active and an EOT branch enabled, the branch will be taken.

HP-IB: Leaves bus in present state.

Serial, BCD, GPIO: Does not affect external lines, so `STATUS` can be used to inspect line states. `RESET` or `ABORTIO` may be necessary after a halt to return handshake lines to the proper state for the next operation.

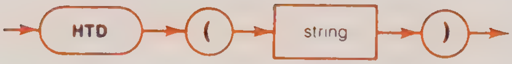
HP-IL:

- Active Controller: If a data transfer is in progress than a Not Ready for Data (NRD) is sent. If the interface is not involved in the transfer then RESUME may be used to continue the transfer.
- Non-controller: Leaves loop in present state.

Related Statements

```
ABORTIO
ON EOT
RESET
```

HTD



HTD(*string*)

Example Statements

```
20 Y=HTD(H$&L$)+A2
40 DISP HTD("F73A")
```

Parameters

string—a string expression that contains the base 16 representation of an integer. Limited to 4 significant characters that must be "0" thru "9" or "A" thru "F".

Action Taken

HTD is a function that returns the value of a base 16 representation contained in the string argument. The argument is a character representation and the result is a numeric quantity.

Related Statements

BTD
DTB$
DTH$
DTO$
OTD

IMAGE

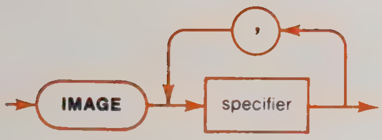


IMAGE *specifier* [, *specifier*] ...

Example Statements

```
10 LBL: IMAGE "Total =",4D.DD
100 IMAGE #,%K,2X,#K
```

Summary of OUTPUT Image Specifiers

Image	Meaning
A	Output one string character
B	Output number as one 8-bit byte
C	Output a comma separator in a number
D	Output one digit character; blank for leading zero
E	Output exponent information; five characters
e	Output exponent information; four characters
K	Output a variable in free-field format
M	Output number's sign if negative, blank if positive
P	Output a period separator in a number
R	Output a European radix point (comma)
S	Output number's sign, plus or minus
W	Output number as two 8-bit bytes (16-bit word)
X	Output one blank
Z	Output one digit character, including leading zeros
" . . . "	Output a literal
#	Suppress end-of-line sequence at end of statement
*	Output one digit character; asterisk for leading zero
.	Output an American radix point (decimal point)
/	Output and end-of-line sequence

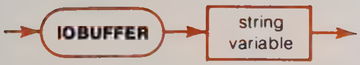
Summary of ENTER Image Specifiers

Image	Meaning
A	Demands one string character
B	Enter number as one 8-bit byte
C	Demand one character for a numeric field; allows commas to be skipped over
D	Demand one character for a numeric field
E	Demand five characters for a numeric field
e	Demand four characters for a numeric field
K	Enter a variable in free-field format
M	Demand one character for a numeric field
S	Demand one character for a numeric field
W	Enter number as two 8-bit bytes (16-bit word)
X	Skip one character
Z	Demand one character for a numeric field
#	Suppress requirement for a line-feed to terminate statement or field
%	Allow EOI to terminate statement or field
*	Demand one character for a numeric field
.	Demand one character for a numeric field
/	Demand a line feed

Related Statements

CONVERT
ENTER...USING
OUTPUT...USING

IOBUFFER



`IOBUFFER` *string variable*

Example Statements

```
10 DIM A$[88]
20 IOBUFFER A$
30 DIM B$[80]
```

Parameters

string variable—the name of a string variable with a dimensioned length 8 characters longer than the desired size of the buffer on the HP-83/85. Length may be the same as is desired on the HP-86/87.

Actions Taken

Eight characters of the string variable are reserved for control of buffer activity on the HP-83/85. On the HP-87 one of the ten IOBUFFER tables is reserved for control of buffer activity.

Buffer empty pointer:

- Initial value = 1. Accessed by Control/Status registers CR0, SR0. Characters are taken from the buffer (by `ENTER` or `TRANSFER`) using the following sequence:

  1. Read character
  2. Increment empty pointer

Buffer fill pointer:

- Initial value = 0. Accessed by Control/Status registers CR1, SR1. Characters are put into the buffer (by `OUTPUT`, `TRANSFER`, or string assignment) using the following sequence:

  1. Increment fill pointer
  2. Store character

Active-out select code:

- Initial value = 0. Accessed by Status register SR3. When active-out select code equals 0, there is no output `TRANSFER` operation active for this buffer. When an output `TRANSFER` is active for this buffer, the active-out select code is set equal to the interface select code that is the destination of the `TRANSFER`.

Active-in select code:

- Initial value = 0. Accessed by Status register SR2. When active-in select code equals 0, there is no input TRANSFER operation active for this buffer. When an input transfer is active for this buffer, the active-in select code is set equal to the interface select code that is the source of the transfer.

Conversion pointers:

- These pointers cannot be accessed from BASIC. When a CONVERT statement to the IOBUFFER is executed, pointers to the appropriate conversion tables are established. **These pointers are initialized by the IOBUFFER statement.** Therefore, execute CONVERT **after** executing IOBUFFER.

Full buffer:

- A buffer is full when the fill pointer equals the dimensioned length of the string (minus eight on the HP-85 or HP-83). Attempting to store data into a full buffer generates a BUFFER error.

Empty buffer:

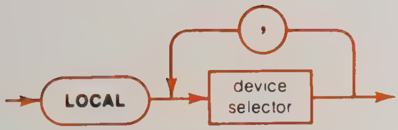
- A buffer is empty when the empty pointer equals the fill pointer plus one. When the buffer becomes empty, the fill pointer is reset to zero, and the empty pointer is reset to one. Active-out and active-in select codes are not affected by the buffer becoming empty; neither are the conversion pointers affected. Old data in the buffer is not lost, but the buffer fill pointer must be modified if you wish to re-access the data in the buffer (so the buffer will "look" full).

Buffer Status Registers		Buffer Control Registers	
Empty pointer	SR0	Empty pointer	CR0
Fill pointer	SR1	Fill pointer	CR1
Active in select code	SR2		
Active out select code	SR3		

Related Statements

CONTROL
CONVERT
ENTER
OUTPUT
STATUS
TRANSFER

LOCAL



`LOCAL` *device selector* [, *device selector*] ...

Example Statements

```
220 LOCAL 7
330 LOCAL 100*S+D1 @ RESUME 7
```

Parameters

device selector—a valid interface select code or a valid combination of interface select code and primary address (see Choosing the Source or Destination in section 1). If multiple device selectors are specified, they must all be on the same interface select code.

Actions Taken

HP-IB:

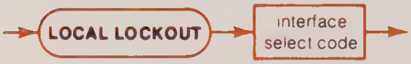
- If device selector is only an interface select code, Remote Enable (REN) is set false. Must be System Controller.
- If device selector contains a primary address, interface addresses specified device(s) and sends Go To Local (GTL) message. Leaves ATN true; use `RESUME` if you wish to set ATN false. Must be Active Controller.
- If device is in REMOTE with LOCAL LOCKOUT set, the device must receive the GTL message or have REN set false to be returned to local (front panel) control.

Serial, BCD, GPIO: Error

HP-IL: Must be active controller.

- If device selector is only an interface select code then Not Remote Enable (NRE) is sent.
- If device selector includes a primary address then Unlisten (UNL), Listen Address n (LADn), and Go To Local (GTL) are sent.
- If device is in REMOTE with LOCAL LOCKOUT set then the device must receive the Go To Local or Not Remote Enable message before it will return to local (front panel) control.

LOCAL LOCKOUT



`LOCAL LOCKOUT` *interface select code*

Example Statements

```
50 LOCAL LOCKOUT 50 @ RESUME 50
100 REMOTE 706,712 @ LOCAL LOCKOUT 7
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

Action Taken

HP-IB and HP-IL:

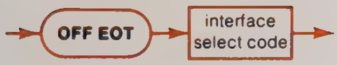
- Must be Active Controller. Sends Local Lockout (LLO) command. (HP-IB leaves ATN true; use `RESUME` if you wish to set ATN false.)
- Local Lockout remains in effect until the Remote Enable (REN) line is set false for HP-IB or, for HP-IL the Not Remote Enable (NRE) command is sent.

Serial, BCD, GPIO: Error

Related Statements

```
LOCAL
REMOTE
```

OFF EOT



`OFF EOT` *interface select code*

Example Statements

```
40 OFF EOT 3
120 IF S>128 THEN OFF EOT S1
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

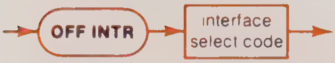
Action Taken

Disables end-of-line branching for termination of a transfer on the specified interface. `OFF EOT` does not cancel a branch permanently. For example, if the transfer has terminated and an `ON EOT` statement is re-executed, the branch will be taken at that time.

Related Statements

```
OFF INTR
OFF TIMEOUT
ON EOT
ON INTR
ON TIMEOUT
```

OFF INTR



`OFF INTR` *interface select code*

Example Statements

```
110 OFF INTR 7
180 IF X THEN OFF INTR S2
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

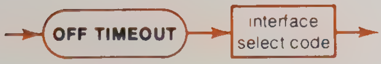
Action Taken

Disables end-of-line branching for interrupts from the specified interface. `OFF INTR` does not cancel a branch permanently. For example, if the interface has interrupted and an `ON INTR` statement is re-executed, the branch will be taken at that time.

Related Statements

```
CONTROL
ENABLE INTR
OFF EOT
OFF TIMEOUT
ON EOT
ON INTR
ON TIMEOUT
```

OFF TIMEOUT



`OFF TIMEOUT` *interface select code*

Example Statements

```
50 OFF TIMEOUT S DIV 100
120 OFF TIMEOUT 7
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

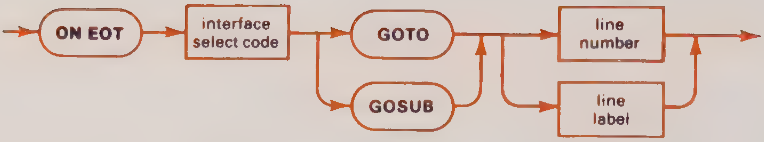
Action Taken

Disables end-of-line branching for occurrence of a timeout on the specified interface. `OFF TIMEOUT` does not cancel a branch permanently. For example, if the interface has timed out and an `ON TIMEOUT` statement is re-executed, the branch will be taken at that time.

Related Statements

```
OFF EOT
OFF INTR
ON EOT
ON INTR
ON TIMEOUT
SET TIMEOUT
```

ON EOT



`ON EOT` *interface select code* `GOTO` *line number*
`GOSUB` *line label*

Example Statements

```
20 ON EOT 7 GOTO SERVICE 7
120 ON EOT S4 GOSUB 1000
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

line number—an integer constant from 1 thru 9999 that specifies a valid line number within the program.

line label—a name of up to 31 characters containing letters, numbers, or underscore symbol, whose first character must be a letter.

Actions Taken

Enables end-of-line branches to the specified line number when a transfer to or from the specified interface is terminated. A pending end-of-line branch from a previous, unserviced transfer termination (for the specified interface select code) is taken immediately. Only one transfer termination per interface select code is retained by the system.

Each interface may have alternate causes for transfer terminations that are user-programmable. Refer to the appropriate interface owner's manual for details about this capability.

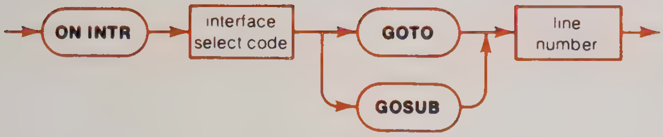
Overrides any previous `ON EOT` statement for the same interface select code.

Related Statements

```
CONTROL
OFF EOT
OFF INTR
OFF TIMEOUT
ON INTR
ON TIMEOUT
STATUS
TRANSFER
```

Also see Branch Precedence Table in section 6.

ON INTR



`ON INTR` *interface select code* `GOTO` *line number*
`GOSUB` *line label*

Example Statements

```
30 ON INTR S1 GOSUB 2000
60 ON INTR 3 GOTO EXIT
150 EXIT: ABORTIO
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

line number—an integer constant from 1 thru 9999 that specifies a valid line number within the program.

Actions Taken

Enables end-of-line branches to the specified line number when an interface interrupt occurs (see `ENABLE INTR`). A pending end-of-line branch from a previous, unserviced interface interrupt (for the specified interface select code) is taken immediately. Only one interrupt per interface select code is retained by the system.

Interrupt causes are specified by either `ENABLE INTR` or `CONTROL` statements. Interrupt causes are interface-dependent; refer to the appropriate interface owner's manual for details.

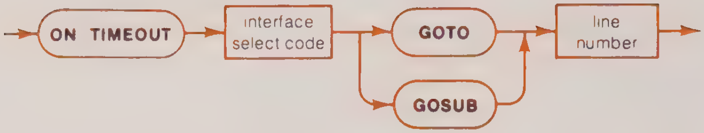
Overrides any previous `ON INTR` statement for the same interface select code.

Related Statements

```
CONTROL
ENABLE INTR
OFF EOT
OFF INTR
OFF TIMEOUT
ON EOT
ON TIMEOUT
```

Also see Branch Precedence Table in section 6.

ON TIMEOUT



`ON TIMEOUT` *interface select code* $\begin{matrix}\texttt{GOTO}\\ \texttt{GOSUB}\end{matrix}$ *line number*

Example Statements

```
20 ON TIMEOUT S3 GOSUB 2500
50 ON TIMEOUT 7 GOTO 320
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

line number—an integer constant from 1 thru 9999 that specifies a valid line number within the program.

Actions Taken

Enables end-of-line branches to the specified line number when an interface timeout occurs (see `SET TIMEOUT`). A pending end-of-line branch from a previous, unserviced interface timeout (for the specified interface select code) is taken immediately. Only one timeout per interface select code is retained by the system.

End-of-line branching for `TIMEOUT` is **not** applicable to the actual data movement portion of a `TRANSFER` (`INTR` or `FHS`) operation. A transfer can timeout if the interface or device cannot be addressed to start the transfer, but there will be no `ON TIMEOUT` branch if the peripheral device stops handshaking in the middle of the transfer.

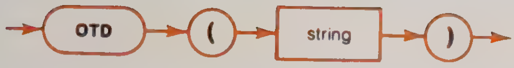
Overrides any previous `ON TIMEOUT` statement for the same interface select code.

Related Statements

```
OFF EOT
OFF INTR
OFF TIMEOUT
ON EOT
ON INTR
SET TIMEOUT
```

Also see Branch Precedence Table in section 6.

OTD



OTD(*string*)

Example Statements

```
80 X=OTD(H$&L$)+A3
110 DISP OTD("177345")
```

Parameters

string—a string expression that contains the base 8 representation of an integer. Limited to 6 significant characters that must be "0" thru "7" (except most significant character must be "0" or "1").

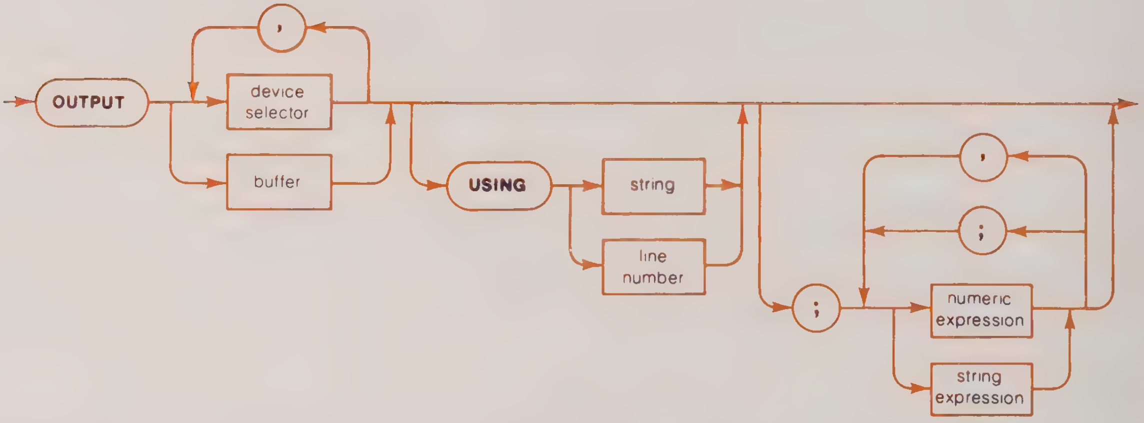
Action Taken

OTD is a function that returns the value of a base 8 representation contained in the string argument. The argument is a character representation and the result is a numeric quantity.

Related Statements

```
DTB$
DTH$
DTO$
BTD
HTD
```

OUTPUT



```
                                                   string
OUTPUT      buffer      [, device selector ...][USING line number [[; expression [, expression]
       device selector                                line label
  [; expression] ... ]
```

Example Statements

```
70 OUTPUT 701 USING Outformat ; X,Y,Z
90 OUTPUT C$ ; N(I) ;Z$
120 OUTPUT 3 USING 30 ; A$
250 OUTPUT 100*S+A USING "#,B" ; N
```

Parameters

device selector—a valid interface select code or a valid combination of interface select code and primary address (see Choosing the Source or Destination). If multiple device selectors are specified, they must all be on the same interface select code. `OUTPUT` allows device selectors 1 and 2 for addressing the internal CRT and printer.

buffer—the name of a string variable that has been declared as an `IOBUFFER`.

string—a string expression that contains a valid set of image specifiers.

line number/label—the line number or label or an `IMAGE` statement that contains a valid set of image specifiers.

expression (string or numeric)—any string expression or numeric expression intended to be output. Expressions may be constants or variables and may be separated by commas or semicolons.

Actions Taken

Outputs bytes to the specified buffer or device(s); bytes may be string or numeric. If a `CONVERT` operation is specified, the conversion is performed immediately before the byte is sent to the interface or buffer.

When `USING` is not specified, and output items are separated by commas, free-field format is used. A free-field output of a string item causes it to be left-justified in a field with no more than 20 trailing blanks. A free-field output of a numeric item causes it to be left-justified in a field of 11, 21, or 32 characters.

When `USING` is not specified, and output items are separated by semicolons, compact format is used. A compact output of a string variable causes it to be sent with no leading or trailing blanks. A compact output of a numeric variable causes it to be sent with one trailing blank and one leading sign character (blank if positive, minus sign if negative).

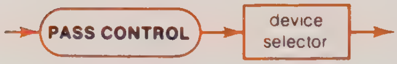
When `USING` is specified, output operations are formatted according to the image specifiers used. Image specifiers may be enclosed in quotes and placed in the `OUTPUT` statement, contained in a string variable named in the `OUTPUT` statement, or placed in an `IMAGE` statement referenced by the `OUTPUT` statement. For detailed information on image specifiers, refer to the `IMAGE` statement in this appendix or see Formatted `OUTPUT` in section 3.

`OUTPUT` sends an end-of-line sequence after the last item in the `OUTPUT` list. This sequence is interface-dependent, can be changed by the `CONTROL` statement, and defaults to carriage return/line feed. This sequence can be suppressed by using `"#"` as the first image specifier. For more detailed information on statement terminators, see Formatted `OUTPUT`. If the `OUTPUT` is to a buffer, a carriage return/line feed is placed in the buffer after the last data byte unless the `"#"` image is used.

Related Statements

```
CONTROL
CONVERT
IMAGE
IOBUFFER
TRANSFER
```

PASS CONTROL



`PASS CONTROL` *device selector*

Example Statements

```
100 PASS CONTROL 100*S+D
250 PASS CONTROL 721 @ ENABLE INTR 7;32
```

Parameters

device selector—a valid interface select code or a valid combination of interface select code and primary address (see Choosing the Source or Destination in section 1).

Actions Taken

HP-IB and HP-IL: Must be Active Controller. Passes Active Controller responsibility to the specified device.

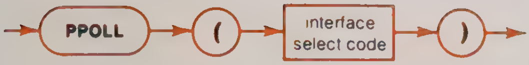
- If device selector is only an interface select code, interface sends the Take Control (TCT) message (and sets ATN false for HP-IB). Be sure that the device receiving control has been addressed to talk before using this form of `PASS CONTROL`.
- If device selector contains a primary address, interface sends the specified device's talk address, sends the TCT message (then sets ATN false for HP-IB).

Serial, BCD, GPIO: Error

Related Statements

```
ABORTIO
ENABLE INTR
ON INTR
REQUEST
RESET
```

PPOLL



`PPOLL(`*interface select code*`)`

Example Statements

```
310 X=PPOLL(7)
620 P9=PPOLL(S0)
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

Actions Taken

HP-IB and HP-IL:

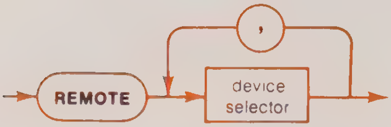
- Must be Active Controller. `PPOLL` is a function that returns the results of a Parallel Poll operation. Sends Identify (IDY) message. Devices capable of responding each assert one bit of the parallel poll response byte.

Serial, BCD, GPIO: Error

Related Statements

`SPOLL`

REMOTE



`REMOTE` *device selector* [, *device selector*] ...

Example Statements

```
50 REMOTE 720
130 REMOTE 100*S+D @ RESUME S
```

Parameters

device selector—a valid interface select code or a valid combination of interface select code and primary address (see Choosing the Source or Destination in section 1). If multiple device selectors are specified, they must all be on the same interface select code.

Actions Taken

HP-IB: Must be System Controller. Puts the bus into remote operation.

- If device selector is only an interface select code, interface sets Remote Enable (REN) true. Devices do not go into remote state until they are addressed to listen.
- If device selector contains a primary address, interface sets REN true, sends Unlisten (UNL) message, then sends the listen address of the specified device(s). `REMOTE` leaves ATN true; use `RESUME` if you wish to set ATN false.

Serial: Error

BCD: Sets partial field separator. Refer to the BCD Interface Owner's Manual for details.

GPIO: Error

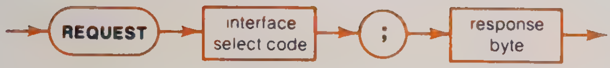
HP-IL: Must be Active Controller.

- If device selector is only an interface select code then the Remote Enable (REN) message is sent.
- If the device selector includes a primary address then Remote Enable (REN), Unlisten (UNL), and Listen Address(es) (LAD) are sent.

Related Statements

```
LOCAL
LOCAL LOCKOUT
RESUME
```

REQUEST



`REQUEST` *interface select code* ; *response byte*

Example Statements

```
50 REQUEST 7 ; 64+4
260 IF T>40 THEN REQUEST S ; 64+X
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

response byte—a numeric expression that evaluates to an integer 0 thru 255.

Actions Taken

HP-IB and HP-IL: Must be non-controller. Sets up a Serial Poll response byte. Sets Service Request (SRQ) true if bit 6 (decimal value 64) of the response byte is set. The response byte is sent to the Active Controller in response to an incoming Serial Poll operation. The Active Controller's Serial Poll operation clears SRQ, which can also be cleared by executing `REQUEST` with bit 6 of the response byte equal to zero.

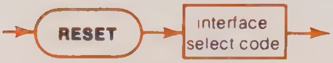
Serial: Sends a BREAK. The BREAK is defined by the response byte. A space (0-state) condition is held for the number of character times specified in the response byte. It is then followed by a mark (1-state) condition for five character times.

BCD, GPIO: Error

Related Statements

```
PASS CONTROL
SPOLL
```

RESET



`RESET` *interface select code*

Example Statements

```
30 RESET 7
300 IF S>128 THEN RESET S3
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

Actions Taken

All interfaces: Performs a hardware reset of the interface, returning it unconditionally to its power-on state. The interface performs a self test (failure causes ERROR 110), and the control registers are set according to the configuration switches on the interface circuit assembly. Resetting an interface with a transfer active and EOT branching enabled causes the branch to be taken.

HP-IB: If System Controller, sends Interface Clear (IFC), then Remote Enable (REN).

Serial: Modem control lines are turned off.

BCD: Data lines are set to high-impedance state, handshake lines are set false, and I/O lines are set to input state.

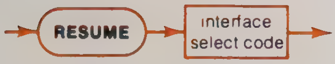
GPIO: Ports A and B are set to high-impedance state, Ports C and D are set to off state, CTL lines are set false, and OUTA and OUTB are set to indicate output.

HP-IL: If the interface is system controller then Interface Clear (IFC), Auto Address Unconfigure (AAU), and Auto Address 1 (AAD1) are sent, followed by Not Remote Enable (NRE) and Remote Enable (REN).

Related Statements

```
ABORTIO
HALT
ON EOT
```

RESUME



`RESUME` *interface select code*

Example Statements

```
110 RESUME 7
190 RESUME S DIV 100
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

Actions Taken

HP-IB: Must be Active Controller (CA = 1). Sets the Attention (ATN) line false. Statements that can leave the ATN line true are: `CLEAR`, `LOCAL`, `LOCAL LOCKOUT`, `REMOTE`, `SEND`, `TRIGGER`.

Serial: The transmitter is enabled. Refer to the *HP 82939A Serial Interface Owner's Manual* for details.

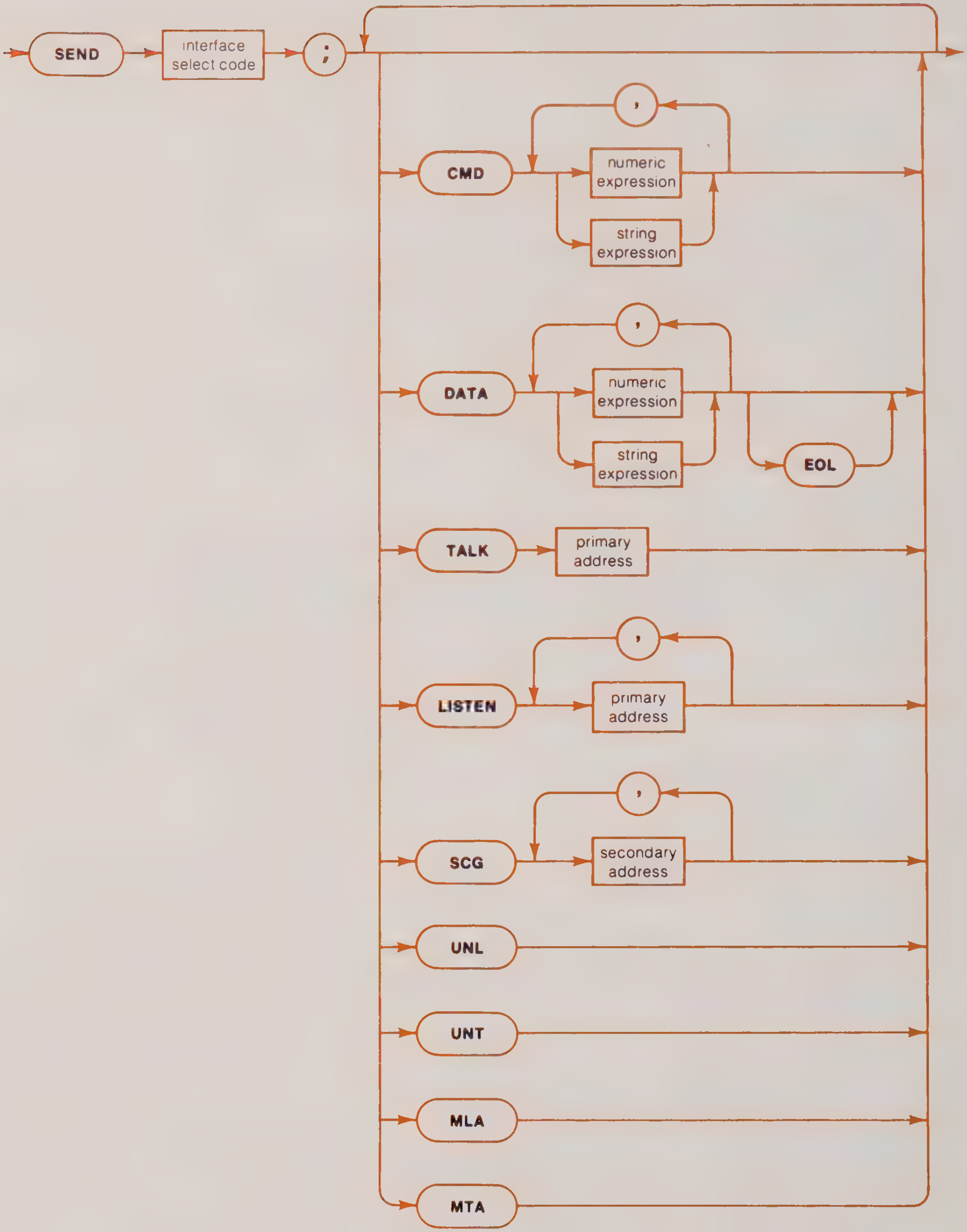
BCD, GPIO: Error

HP-IL: Must be active controller. The Send Data (SDA) message is sent if a transfer is not already in progress.

Related Statements

```
CONTROL
HALT
SEND
```

SEND


SEND
interface select code
;
CMD
,
numeric expression
string expression
DATA
,
numeric expression
string expression
EOL
TALK
primary address
LISTEN
,
primary address
SCG
,
secondary address
UNL
UNT
MLA
MTA

`SEND` *interface select code* ; [[`CMD` *list*][`DATA` *list* [`EOL`]][`TALK` *primary address*][`LISTEN` *primary address* [, *primary address*] ...][`SCG` *secondary address* [, *secondary address*] ...][`UNL`] [`UNT`][`MLA`][`MTA`] ...]

Example Statements

```
100 SEND 7 ; CMD "U?%" DATA "Hello"
200 SEND 7 ; CMD A$ SCG 14,18 DATA X$
300 SEND S ; MTA UNL LISTEN 6,14 CMD P,R SCG 6
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

list—a list of numeric or string expressions, separated by commas.

primary address—a numeric expression that evaluated to an integer 0 thru 31.

secondary address—a numeric expression that evaluates to an integer 0 thru 31.

Actions Taken

HP-IB: When sending any commands (`CMD`, `TALK`, `LISTEN`, `SCG`, `UNL`, `UNT`, `MLA`, `MTA`), the interface must be Active Controller. The ATN line is set true while sending commands. The ATN line is set false while sending `DATA`, even if no actual data is sent (i.e., `DATA " "`).

- `CMD`: Commands: send list of 8-bit expressions with ATN true. Primary commands have a bit pattern = X00CCCCC, where X = don't care, C = bits of command (decimal 0 thru 31). `SEND CMD` can be used to create odd parity on commands, if necessary.
- `DATA`: Send list of numeric or string expressions with ATN false. Any 8-bit pattern may be sent. If EOL is specified, the interface's end-of-line character sequence is sent following data (Control registers 17-thru 23).
- `TALK`: Send device's Talk Address (TAD), decimal 0 thru 31.
- `LISTEN`: Send device's Listen Address (LAD), decimal 0 thru 31.
- `SCG`: Secondary Command Group: Send secondary address to device.
- `UNL`: Send Unlisten command (UNL). Numeric value sent is 63; ATN is true.
- `UNT`: Send Untalk command (UNT). Numeric value sent is 95; ATN is true.
- `MLA`: Send My Listen Address (MLA). This is the listen address of the interface. Factory setting = 53.
- `MTA`: Send My Talk Address (MTA). This is the talk address of the interface. Factory setting = 85.

Serial: The only form that can be sent is `DATA`.

- `DATA`: Sends list of numeric or string expressions. If EOL is specified, the interface's end-of-line character sequence is sent (control registers 17 thru 23).

BCD: See BCD Owner's Manual for details:

- CMD: Primary addresses 0 thru 6 set partial field specifier.
- DATA: Lower 4 bits of data bytes are sent; control characters, spaces, and commas are ignored. If EOL is specified, data format checking is enabled.
- LISTEN, TALK: Primary addresses 0 thru 6 set partial field specifier.
- SCG: Error.
- UNL, UNT, MLA, MTA: Ignored.

GPIO: See GPIO Owner's Manual for details.

- CMD: Primary addresses 0 thru 15 select port configuration. Device Clear command pulses RESA and RESB. Selected Device Clear pulses RESA or RESB according to the most recent primary address.
- DATA: Send list of numeric or string expressions. Data is sent as 8-bit bytes. If EOL is specified, the interface's end-of-line character sequence is sent (control registers 17 thru 23).
- LISTEN, TALK: Primary addresses 0 thru 15 select port configuration.
- SCG: Error.
- UNL, UNT, MLA, MTA: Ignored.

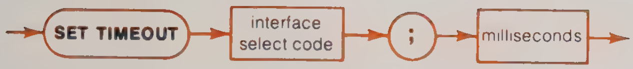
HP-IL: When sending commands (CMD, TALK, LISTEN, SCG, UNL, UNT, MLA, MTA) the interface must be active controller.

- CMD: Sends a list of 8 bit expressions as command frames.
- DATA: Sends a list of 8 bit expressions as data frames. If EOL is specified then the interface end-of-line sequence is sent following the data.
- TALK: Sends a device's Talk Address, decimal 0 to 31.
- LISTEN: Sends device Listen Address(es), decimal 0 to 31.
- SCG: Sends a secondary address frame, decimal 0 to 31.
- UNL: Sends an Auto Address sequence and the Unlisten command frame.
- UNT: Sends an Untalk command frame.
- MLA: Addresses the interface to Listen.
- MTA: Sends the Talk Address of the interface.

Related Statements

OUTPUT

SET TIMEOUT



`SET TIMEOUT` *interface select code* ; *milliseconds*

Example Statements

```
100 SET TIMEOUT S0 ; X*1000
280 ON TIMEOUT 7 GOTO 550 @ SET TIMEOUT 7 ; 4500
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

milliseconds—a numeric expression that evaluates to an integer 0 thru 32,767.

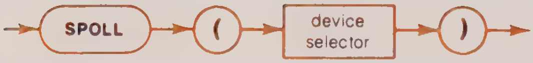
Action Taken

Establishes an approximate time limit (in milliseconds) that the interface will wait to complete a handshake with its peripheral device. If the specified time limit is exceeded and `ON TIMEOUT` end-of-line branching is enabled, the branch is taken. If no `ON TIMEOUT` is currently in effect, there is **no** indication that a timeout has occurred until an `ON TIMEOUT` is subsequently executed.

Related Statements

```
OFF TIMEOUT
ON TIMEOUT
```

SPOLL



`SPOLL(`*device selector*`)`

Example Statements

```
50 P=SPOLL(S4)
250 IF SPOLL(701)>63 THEN GOTO 750
```

Parameters

device selector—a valid interface select code or a valid combination of interface select code and primary address (see "Choosing the Source or Destination").

Actions Taken

HP-IB:

- Conducts a Serial Poll of a device on the bus and returns the device's status byte. If bit 6 of the status byte is set (decimal value 64), it indicates that the device is requesting service (asserting SRQ).
- If device selector is only an interface select code, interface sends Serial Poll Enable (SPE), sets ATN false, receives the status byte, sends Serial Poll Disable (SPD), then sends Untalk (UNT).
- If device selector contains a primary address, interface sends Unlisten (UNL), My Listen Address (MLA), devices Talk Address (TAD), Serial Poll Enable (SPE), then sets ATN false. It receives the status byte, sends Serial Poll Disable (SPD), then sends Untalk (UNT).

Serial, BCD, GPIO: Error.

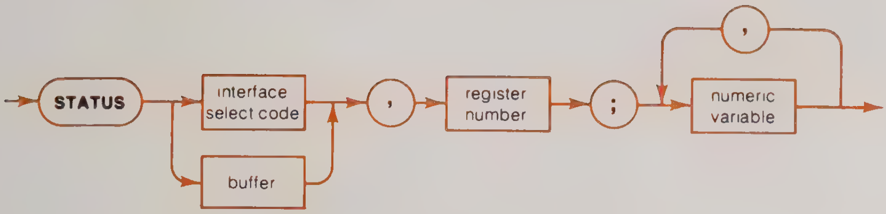
HP-IL: Must be active controller.

- `SPOLL` returns the first byte received in response to a serial poll of a device.
- If the device selector is just an interface select code then the interface sends the Send Status (SST) message, then waits to receive a data byte followed by end of transmission (EOT) and then sends (UNT).
- If the device selector includes a primary address then unlisten (UNL), my talk address (MTA), the device's talk address (TAD) and send status (SST) are sent. The data byte is received followed by end of transmission (EOT) and then untalk (UNT) is sent.

Related Statements

`PPOLL`

STATUS



`STATUS` *buffer*
interface select code , *register number* ; *numeric variable* [, *numeric variable*] ...

Example Statements

```
20 STATUS 7,0 ; C0,C1,C2,C3,C4
70 STATUS S1,5 ; .S1-5*
```

Parameters

interface select code—a numeric expression that evaluates to an integer 3 thru 10.

buffer—the name of a string variable that has been declared as an `IOBUFFER`.

register number—a numeric expression that evaluates to an integer 0 thru 15. Must specify a valid status register for the selected interface.

numeric variable—any numeric variable intended as a destination for the status information.

Actions Taken

Reads one or more status register(s) and assigns the value(s) to the specified variable(s). When more than one variable is specified, consecutive status registers are read, starting at the specified register number. Status values returned are integers 0 thru 255.

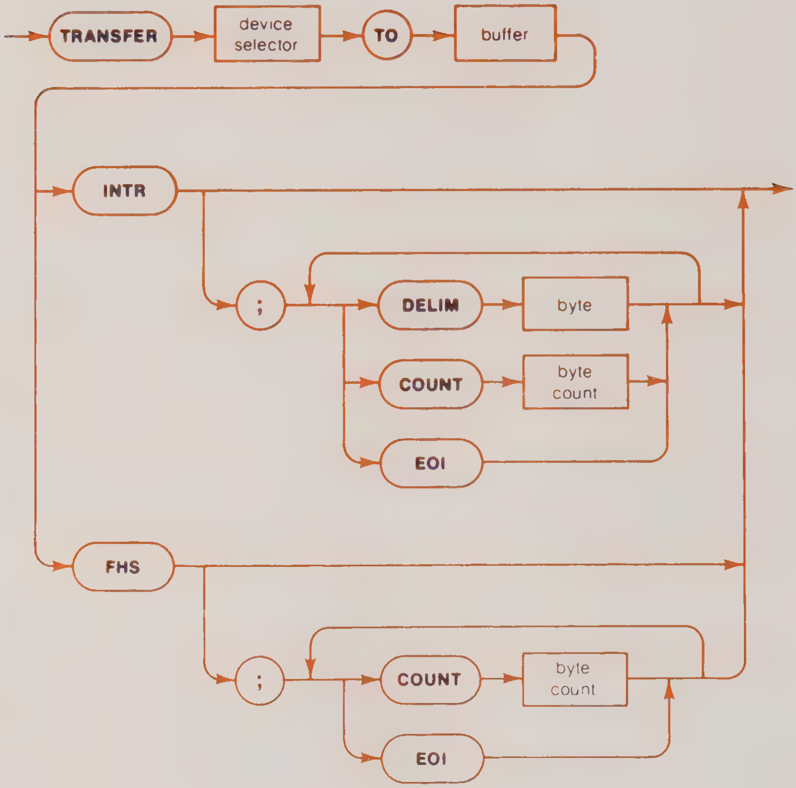
Related Statements

```
ASSERT
CONTROL
ENABLE INTR
IOBUFFER
```

---

*Labels are allowed on the HP-87.

TRANSFER (in)


TRANSFER
device selector
TO
buffer
INTR
;
DELIM
byte
COUNT
byte count
EOI
FHS
;
COUNT
byte count
EOI

TRANSFER *device selector* TO *buffer* INTR [; [COUNT *byte count*][DELIM *byte*][EOI]]

TRANSFER *device selector* TO *buffer* FHS [; [COUNT *byte count*][EOI]]

Example Statements

```
100 TRANSFER 706 TO B$ INTR
200 TRANSFER 100*S+D TO B$ INTR ; COUNT 80 DELIM 10 EOI
300 TRANSFER 3 TO A$ FHS ; COUNT 16
```

Parameters

device selector—a valid interface select code or a valid combination of interface select code and primary address (see "Choosing the Source or Destination").

buffer—the name of a string variable that has been declared as an `IOBUFFER`.

byte count—a numeric expression that evaluates to an integer 0 thru 32 767. Specifies maximum number of bytes to be input.

byte—a numeric expression that evaluates to an integer 0 thru 255. Specifies the ASCII value of a character which can terminate the transfer.

Actions Taken

Takes data bytes from the specified device and places them into the specified buffer. Characters are placed into the buffer according to the buffer fill pointer. The transfer terminates when the buffer is full or when the first one of the specified terminating conditions is met. The interface may also have a programmable terminating condition; refer to the appropriate interface owner's manual for details. Specifying `COUNT` sets a maximum limit on the number of characters to be transferred. `DELIM` specifies the numeric value of a character that can terminate the transfer. Specifying EOI (End or Identify) allows the transfer to terminate when an interface-dependent "END" signal is detected (such as the EOI line on HP-IB). The terminating condition for buffer full is always in effect. If an `ON EOT` branch is enabled, the branch is taken when the transfer terminates.

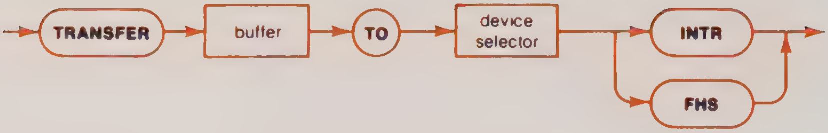
If `INTR` (Interrupt) is specified, the interface is automatically enabled to interrupt the computer each time it is ready with a new character. The transfer continues to completion even though program execution may have stopped. A `WARNING 101` is issued if the program stops with a transfer still active. Be certain that the transfer has terminated (use `RESET`, `HALT`, or `ABORTIO`) before attempting to modify the program. This transfer type (under ideal conditions) is capable of a maximum data transfer rate of about 400 bytes per second.

If `FHS` (Fast Handshake) is specified, the interface and computer are dedicated to the transfer until it is complete. No interrupts of keypresses (not even the RESET key) are detected until the transfer terminates. If the computer "locks up" on a `FHS TRANSFER`, only a power-on or special interface-specific termination (i.e., Interface Clear on HP-IB) can return the computer to its normal state. This transfer type (under ideal conditions) is capable of data transfer rates in excess of 20 000 bytes per second.

Related Statements

```
ABORTIO
CONTROL
ENTER
HALT
IOBUFFER
ON EOT
RESET
STATUS
```

TRANSFER (out)



INTR
TRANSFER *buffer* TO *device selector*
FHS

Example Statements

```
150 TRANSFER B$ TO 721 INTR
280 OUTPUT B$ USING "#,K" ; D$ @ TRANSFER B$ TO 9 FHS
400 ON EOT S1 GOSUB 660 @ TRANSFER X$ TO S1 INTR
```

Parameters

buffer—the name of a string variable that has been declared as an IOBUFFER.

device selector—a valid interface select code or a valid combination of interface select code and primary address (see Choosing the Source or Destination in section 1 of this manual). With HP-IB, and only a select code specified, a multiple device transfer may be obtained.

Actions Taken

Takes data bytes from the specified buffer and sends them to the specified device. Data is taken from the buffer according to the buffer empty pointer. If the device selector contains a primary address, addressing is performed prior to sending the first byte. The interface's programmable end-of-line sequence is sent after the last byte from the buffer has been sent. Note that the buffer may contain an additional carriage-return/line-feed placed there by an OUTPUT statement. The transfer terminates when the buffer is empty. If ON EOT branching is enabled, the branch is taken when the transfer terminates.

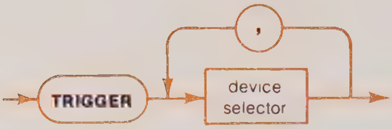
If INTR (Interrupt) is specified, the interface is automatically enabled to interrupt the computer each time it is ready for a new character. The transfer continues to completion even though program execution may have stopped. A WARNING 101 is issued if the program stops with a transfer still active. Be certain that the transfer has terminated (use STATUS, RESET, HALT, or ABORTIO) before attempting to modify the program. This TRANSFER type (under ideal conditions) is capable of a maximum data transfer rate of about 400 bytes per second.

If FHS (Fast Handshake) is specified, the interface and computer are dedicated to the transfer until it is complete. No interrupts or keypresses (not even the RESET key) are detected until the transfer terminates. If the computer "locks up" on a FHS TRANSFER, only a power-on or special interface-specific termination (i.e., Interface Clear on HP-IB or HP-IL) can return the computer to its normal state. This transfer type (under ideal conditions) is capable of data transfer rates in excess of 20 000 bytes per second.

Related Statements

```
ABORTIO
CONTROL
HALT
IOBUFFER
ON EOT
OUTPUT
RESET
STATUS
```

TRIGGER



`TRIGGER` *device selector* [, *device selector*] ...

Example Statements

```
70 TRIGGER 706,715 @ ENTER 706 ; V1
190 TRIGGER S1
```

Parameters

device selector—a valid interface select code or a valid combination of interface select code and primary address (see Choosing the Source or Destination in section 1). If multiple devices are selected, they must all be on the same interface select code.

Actions Taken

HP-IB: Must be Active Controller. Sends the Group Execute Trigger command (GET).

- If device selector is only an interface select code, interface sends the GET command. Those devices already addressed to listen respond to the GET command.
- If device selector contains a primary address, interface sends Unlisten (UNL), then the Listen Address (LAD) of the specified device(s). Sends the GET command (HP-IB leaves ATN true; use `RESUME` if you wish to set ATN false).

Serial, BCD, GPIO: Error.

HP-IL: Must be active controller. If the device selector is only an interface select code then the group execute trigger (GET) message is sent. If the device selector includes a primary address then unlisten (UNL), listen address (LADn) and group execute trigger are sent.

Related Statements

```
RESUME
SEND
```

Appendix B

Maintenance, Service, and Warranty

Maintenance

The I/O ROM does not require maintenance. However, there are several areas of caution that you should be aware of. They are:

WARNING: Do not place fingers, tools, or other foreign objects into the plug-in ports. Such actions may result in minor electrical shock hazard and interference with some pacemaker devices. Damage to plug-in port contacts and the computer's internal circuitry may also result.

CAUTION: Always switch off the computer and any peripherals involved when inserting or removing modules. Use only plug-in modules designed by Hewlett-Packard specifically for the HP Series 80 Personal Computer that you are using. Failure to do so could damage the module, the computer, or the peripherals.

CAUTION: If a module or ROM drawer jams when inserted into a port, it may be upside down or designed for another port. Attempting to force it may damage the computer or the module. Remove the module carefully and reinsert it.

CAUTION: Handle the plug-in ROMs very carefully while they are out of the ROM drawer. Do not insert any objects in the contact holes on the ROM. Always keep the protective cap in place over the ROM contacts while the ROM is not plugged into the ROM drawer. Failure to observe these cautions may result in damage to the ROM or ROM drawer.

For instructions on how to insert and remove the ROM and ROM drawer, please refer to the *ROM Drawer Instruction Sheet* or the HP Series 80 owner's manuals.

Service

If at any time you suspect that the ROM drawer or I/O ROM may be malfunctioning, do the following:

1. Turn the computer and all peripherals off. Disconnect all peripherals and remove the ROM drawer from the computer ports. Turn the computer back on. If the computer does not respond or displays `Error 23 : SELF TEST`, the computer requires service.

2. Turn the computer off. Install the ROM drawer, with the I/O ROM installed, into any port. Turn the computer back on.

   - If `Error 112 : I/O ROM` is displayed, indicating that the ROM is not operating properly, turn the computer off and try the ROM in another ROM drawer slot. This will help you determine if particular slots in the ROM drawer are malfunctioning, or if the ROM itself is malfunctioning.
   - If the cursor does not appear, the system is not operating properly. To help determine what is causing the improper operation, repeat step 2 with the ROM drawer installed in a different port, both with the I/O ROM installed in the ROM drawer and with the I/O ROM removed from the ROM drawer.

3. Refer to How to Obtain Repair Service for information on how to obtain repair service for the malfunctioning device.

Warranty Information

The complete warranty statement is included in the information packet shipped with your ROM. Additional copies may be obtained from any authorized HP dealer, or the HP sales and service office where you purchased your system.

If you have questions concerning the warranty, and you are unable to contact the authorized HP sales and service office where you purchased your computer, please contact:

In the U.S.:

Hewlett-Packard
Corvallis Division Customer Support
1000 N.E. Circle Blvd.
Corvallis, OR 97330
Tel. (503) 758-1010
Toll Free Number: (800) 547-3400 (except
in Oregon, Hawaii and Alaska).

In Europe:

Hewlett-Packard S.A.
7, rue du Bois-du-lan
P. O. Box
CH-1217 Meyrin 2
Geneva
Switzerland
Tel. (22) 82 70 00

Other Countries:

Hewlett-Packard Intercontinental
3495 Deer Creek Rd.
Palo Alto, California 94304
U.S.A.
Tel. (415) 857-1501

How to Obtain Repair Service

Not all Hewlett-Packard facilities offer service for the HP Series 80 Personal Computer and its peripherals. For information on service in your area, contact your nearest authorized HP dealer or the nearest Hewlett-Packard sales and service office.

If your system malfunctions and repair is required, you can help assure efficient servicing by having the following items with your unit(s) at the time of service:

1. A description of the configuration of the computer, exactly as it was at the time of malfunction, including any plug-in modules, tape cartridges or other accessories.
2. A brief description of the malfunction symptoms for service personnel.
3. Printouts or any other materials that illustrate the problem area.
4. A copy of the sales slip or other proof of purchase to establish the warranty coverage period.

General Shipping Instructions

Should you ever need to ship any portion of your computer system, be sure it is packed in a protective package (use the original case), to avoid in-transit damage. Hewlett-Packard suggests that the customer always insure shipments.

If you happen to be outside of the country where you bought your computer or peripheral, contact the nearest authorized HP Series 80 dealer or the local Hewlett-Packard office. All customs and duties are your responsibility.

Appendix C

Error Messages

Error Message	Meaning	Possible Corrective Action
101 XFR	This is only a warning. It is issued when a program is paused with an I/O transfer still active. Do not attempt to modify a program when a transfer is active.	Before you modify or rerun the program, stop all active transfers with a RESET, HALT, or ABORTIO instruction; or press the RESET key.
110 I/O CARD	An interface has failed self-test. This indicates a probable hardware problem.	ERRSC can be used to determine which interface has failed. Try recycling the power (turn computer off, then back on again). If the interface still fails, contact the authorized dealer or the HP sales and service office from which you purchased your computer.
111 I/O OPER	The I/O operation attempted is not valid with the type of interface being used. Some examples are: specifying a status or control register that does not exist, using a primary address with a Serial interface, or using an I/O statement that is not defined for the interface being used.	ERRL can be used to identify the improper statement. Check this statement in the Syntax Reference section to determine if it is defined for the interface being used. If the statement is valid, check the appropriate interface owner's manual to get details on the proper mode or configuration required for the statement used.
112 I/O ROM	The I/O ROM has failed the checksum self-test. This indicates a probable hardware problem.	Try recycling the power (turn the computer off, then back on again). If the error keeps recurring, contact the authorized dealer or the HP sales and service office from which your purchased your computer.
113	An interface-dependent error. HP-IB: The statement used requires the interface to be system controller. Serial: UART receiver overrun; data has been lost. BCD: Attempting to put the interface into an illegal mode. GPIO: An odd number of bytes was transferred when the interface was configured for 16-bit words. HP-IL: The Take Control (TCT) message was ignored by the device.	ERRSC can be used to determine the source of the error. Refer to the appropriate interface owner's manual to get details on the error and possible corrective actions.

Error Message	Meaning	Possible Corrective Action
114	An interface-dependent error. HP-IB and HP-IL: The statement used requires the interface to be active controller. Serial: Receiver buffer overrun; data has been lost. BCD: Port 10 not currently available. GPIO: FHS TRANSFER aborted by STO.	ERRSC can be used to determine the source of the error. Refer to the appropriate interface owner's manual to get details on the error and possible corrective actions.
115	An interface-dependent error. HP-IB and HP-IL: The statement used requires the interface to be addressed to talk. Serial: Automatic disconnect forced. BCD: FHS TRANSFER aborted by FLGB. GPIO: Interface configuration does not allow an output enable or output operation on Port A or Port B.	ERRSC can be used to determine the source of the error. Refer to the appropriate interface owner's manual to get details on the error and possible corrective actions.
116	An interface-dependent error. HP-IB and HP-IL: The statement used requires the interface to be addressed to listen. Serial: This error number not currently used. BCD: Data direction mismatch on current operation. GPIO: Cannot start operation because handshake CTL line is not in proper state.	ERRSC can be used to determine the source of the error. Refer to the appropriate interface owner's manual to get details on the error and possible corrective actions.
117	An interface-dependent error. HP-IB and HP-IL: The statement used requires the interface to be non-controller. Serial: This error number not currently used. BCD: Interface command has been directed to a non-existent field. GPIO: This error number not currently used.	ERRSC can be used to determine the source of the error. Refer to the appropriate interface owner's manual to get details on the error and possible corrective actions.
118	An interface-dependent error. HP-IB: This error number not currently used. Serial: This error number not currently used. BCD: Cannot start operation because CTL line is not in the proper state. GPIO: This error number not currently used. HP-IL: Protocol violation or loop transmission error.	ERRSC can be used to determine the source of the error. Refer to the appropriate interface owner's manual to get details on the error and possible corrective actions.

Error Message	Meaning	Possible Corrective Action
119	An interface-dependent error. HP-IB: This error number not currently used. Serial: This error number not currently used. BCD: Data format does not match the mode of the interface. GPIO: This error number not currently used. HP-IL: Addressed talker ignored the start of transmission (SOT) frame.	ERRSC can be used to determine the source of the error. Refer to the appropriate interface owner's manual to get details on the error and possible corrective actions.
123 NO ";"	Syntax error. A semicolon delimiter was expected in the statement.	Put the semicolon where it belongs.
124 ISC	Either the interface select code specified is out of range, or there is no interface present set to the specified select code. Interface select codes must be in the range of 3 thru 10. Select codes 1 (CRT) and 2 (internal printer) are allowed for OUTPUT statements only.	Be sure that the interface select code is within the proper range. Pay special attention to variables that are used to hold interface select codes. If the interface select code is OK, be sure that the interface is plugged in properly. Finally, check the switch settings on the interface. (Someone might have changed them last weekend.)
125 ADDR	The primary address specified is improper. Only addresses 00 thru 31 are allowed, but not all interfaces use this entire range.	Be sure that the primary address is within the proper range. Pay special attention to variables that are used to hold addresses or device selectors.
126 BUFFER	Four possible buffer problems: (1) The string variable specified has not been declared as an IOBUFFER. (2) Attempting to ENTER from a buffer which is out of data. (3) Attempting to OUTPUT to a buffer which is already full. (4) Attempting an output TRANSFER with an empty buffer.	Be sure you have included the necessary IOBUFFER statement. Check the logical flow of your program (in what order the statements are executed). Buffer contents can be examined at any time by simply printing or displaying the string variable being used as the buffer. If this doesn't provide enough information, the buffer pointers can be examined with the STATUS statement.
127 NUMBER	An incoming character sequence does not constitute a valid number, or a number being output requires three exponent digits and an "e" format was specified.	If the error is from an output operation, check the magnitude of the number and the format used. If the error is from an input operation, there are many possible causes. Here are some things to look for: more than 255 leading non-numeric characters, unexpected spaces in a character stream when a character-count format is used, punctuation sequences that include potentially numeric characters used in an order that is numerically meaningless.

Error Message	Meaning	Possible Corrective Action
128 EARLY TERM	A buffer was emptied before all the ENTER fields were satisfied, or a field terminator was encountered before the specified character count was reached.	Check your incoming character stream, ENTER list, and image specifiers.
129 VAR TYPE	The type (string or numeric) of a variable in an ENTER list does not match with the image specified for that variable.	Check your ENTER list and image specifiers.
130 NO TERM	A required terminator was not received from an interface or buffer during an ENTER statement. Remember that there is a default requirement for a line-feed statement terminator.	Check your incoming character stream, ENTER list, and image specifiers.

Appendix D

ASCII Character Set

EQUIVALENT FORMS Char.	Binary	Octal	Dec	EQUIVALENT FORMS Char.	Binary	Octal	Dec	EQUIVALENT FORMS Char.	Binary	Octal	Dec	EQUIVALENT FORMS Char.	Binary	Octal	Dec
@^{c}	00000000	000	0	SPACE	00100000	040	32	@	01000000	100	64	` KEY LABEL^s	01100000	140	96
A^c	00000001	001	1	!	00100001	041	33	A	01000001	101	65	a	01100001	141	97
B^c	00000010	002	2	"	00100010	042	34	B	01000010	102	66	b	01100010	142	98
C^c	00000011	003	3	#	00100011	043	35	C	01000011	103	67	c	01100011	143	99
D^c	00000100	004	4	$	00100100	044	36	D	01000100	104	68	d	01100100	144	100
E^c	00000101	005	5	%	00100101	045	37	E	01000101	105	69	e	01100101	145	101
F^c	00000110	006	6	&	00100110	046	38	F	01000110	106	70	f	01100110	146	102
G^c	00000111	007	7		00100111	047	39	G	01000111	107	71	g	01100111	147	103
H^c	00001000	010	8	(	00101000	050	40	H	01001000	110	72	h	01101000	150	104
I^c	00001001	011	9		00101001	051	41	I	01001001	111	73	i	01101001	151	105
J^c	00001010	012	10	*	00101010	052	42	J	01001010	112	74	j	01101010	152	106
K^c	00001011	013	11	+	00101011	053	43	K	01001011	113	75	k	01101011	153	107
L^c	00001100	014	12	,	00101100	054	44	L	01001100	114	76	l	01101100	154	108
M^c	00001101	015	13	−	00101101	055	45	M	01001101	115	77	m	01101101	155	109
N^c	00001110	016	14		00101110	056	46	N	01001110	116	78	n	01101110	156	110
O^c	00001111	017	15		00101111	057	47	O	01001111	117	79	o	01101111	157	111
P^c	00010000	020	16	0	00110000	060	48	P	01010000	120	80	p	01110000	160	112
Q^c	00010001	021	17	1	00110001	061	49	Q	01010001	121	81	q	01110001	161	113
R^c	00010010	022	18	2	00110010	062	50	R	01010010	122	82	r	01110010	162	114
S^c	00010011	023	19	3	00110011	063	51	S	01010011	123	83	s	01110011	163	115
T^c	00010100	024	20	4	00110100	064	52	T	01010100	124	84	t	01110100	164	116
U^c	00010101	025	21	5	00110101	065	53	U	01010101	125	85	u	01110101	165	117
V^c	00010110	026	22	6	00110110	066	54	V	01010110	126	86	v	01110110	166	118
W^c	00010111	027	23	7	00110111	067	55	W	01010111	127	87	w	01110111	167	119
X^c	00011000	030	24	8	00111000	070	56	X	01011000	130	88	x	01111000	170	120
Y^c	00011001	031	25	9	00111001	071	57	Y	01011001	131	89	y	01111001	171	121
Z^c	00011010	032	26		00111010	072	58	Z	01011010	132	90	z	01111010	172	122
[^{c}	00011011	033	27	;	00111011	073	59	[	01011011	133	91	π (/)^s	01111011	173	123
\$^c$	00011100	034	28		00111100	074	60	\	01011100	134	92	|	01111100	174	124
]^c	00011101	035	29	=	00111101	075	61	]	01011101	135	93	→ (−)^s	01111101	175	125
∧^{c}	00011110	036	30		00111110	076	62	^	01011110	136	94	Σ (✱)^s	01111110	176	126
_^{c}	00011111	037	31	?	00111111	077	63	_	01011111	137	95	⊢ (+)^s	01111111	177	127

Notes

Notes

Notes

HEWLETT
PACKARD

Personal Computer Division
1010 N.E. Circle Blvd., Corvallis, OR 97330 U.S.A.

Reorder Number
00087-90121

Printed in U.S.A. 1/83
00087-90263

[illegible]

HEWLETT-PACKARD

HP-IB

INSTALLATION AND THEORY OF OPERATION MANUAL

HEWLETT
PACKARD

HP 82937A

HP-IB Installation and Theory of Operation Manual

October 1980

82937-90007 Rev. B 10/80

Printed in U.S.A.

© Hewlett-Packard Company 1980

Contents

Section 1: General Information ... 1
- Introduction ... 1
  - I/O ROMs ... 1
  - Technical Specifications ... 1
- Bus Functions ... 4
- HP-IB Messages ... 6

Section 2: Installation ... 9
- Unpacking and Inspection ... 9
- Installation ... 9
  - Disassembly ... 9
  - Switch Settings ... 11
  - Parallel Poll Jumper ... 18
  - Installing the Interface and Connecting Peripherals ... 20
- Interconnecting Cables ... 22
  - Cable Length Description ... 23
  - Metric Conversion Kit ... 23
- Functional Test ... 24

Section 3: Theory of Operation ... 29
- Introduction ... 29
  - Translator IC Description ... 29
  - Select Codes ... 29
  - Translator I/O Registers ... 31
  - Interrupts ... 32
  - HP-85 I/O Backplane Lines ... 34
  - HP-IB Bus Lines ... 35
  - Data Transfer ... 37
  - Polling ... 38
  - Serial Poll ... 38
  - Parallel Poll ... 39
  - Switch Buffer ... 39
  - External Control Latch ... 39
  - Bi-directional Bus Transceivers ... 40
  - 8049 Microcomputer ... 41

Figures

2-1 Disassembly ... 10
2-2 Select Code Switches ... 13
2-3 Talk/Listen Address Switches ... 15
2-4 System Controller Switch ... 17
2-5 Removing Cable Connector ... 19
2-6 Parallel Poll Jumper Wire Placement ... 19
2-7 Installing the Interface ... 20
2-8 Bus Cables ... 22
3-1 Translator IC Block Diagram ... 30
3-2 Translator Control Register Bit Assignments ... 31
3-3 Translator Status Register Bit Assignments ... 31
3-4 Interrupt Flow Chart ... 33
3-5 HP-85 I/O Slot Connector Pinout ... 35
3-6 Data Transfer Timing Diagram ... 37
3-7 μC Status Registers ... 42
3-8 μC Control Registers ... 43
3-9 Schematic Diagram ... 45

Tables

1-1 HP-IB Signal Lines ... 2
1-2 Logic Level Requirements ... 2
1-3 Interface Functions and Allowed Capability ... 5
1-4 HP-IB Messages ... 6
2-1 Parallel Poll Jumper Placement ... 18
3-1 HP-85 I/O Backplane Lines ... 34

3-2 8049 Microcomputer Pin Assignment 41
3-3 Replaceable Parts 44
3-7 μC Status Registers 42
3-8 μC Control Registers 43

Section 1

General Information

Introduction

The 82937A Interface connects the HP-85 Personal Computer to the HP Interface Bus. The interface conforms to IEEE Standard 488-1978 and supports a wide variety of operations.

This manual provides general information about the interface, how to install it and a theory of operation section.

I/O ROMs

This interface requires one of three available ROMs to perform input-output operations. The type of ROM used determines which bus operations can be performed. The available ROMs include:

- I/O ROM (P/N 00085-15003)
- Plotter/printer ROM (P/N 00085-15002)
- Mass storage ROM (P/N 00085-15001)

The I/O ROM is a general purpose ROM and is used in a wide variety of applications. The plotter/printer ROM simplifies interfacing plotter and printer devices and the mass storage ROM was developed to interface a floppy disc.

These ROMs fit into a ROM drawer which plugs into any of the four I/O slots on the HP-85. Ensure that you have the proper ROM for your system.

Technical Specifications

Pertinent specifications of the interface are listed in this section. If complete details of HP-IB electrical, mechanical and timing requirements are desired, refer to IEEE Standard 488-1978.

Select Code Switches

The interface is preset at the factory to select code ''7''. Switches located on the interface circuit board allow the select code to be changed. The procedure for changing the switch settings is discussed in Section 2.

Parallel Poll Jumper

A provision which allows the interface to respond to a parallel poll is implemented with a jumper wire on the interface circuit board. The interface is delivered with the jumper wired to assign data line DIO1 as the parallel poll response line. The procedure for changing the parallel poll response line is discussed in Section 2.

Bus Signal Lines

The bus consists of 16 signal lines. These are listed in Table 1-1. A more detailed description of these lines is given in Section 3.

Table 1-1. HP-IB Signal Lines

DIO1	Data Input/Output 1
•	•
•	•
•	•
DIO8	Data Input/Output 8
DAV	Data Valid
NRFD	Not Ready for Data
NDAC	Not Data Accepted
IFC	Interface Clear
ATN	Attention
SRQ	Service Request
REN	Remote Enable
EOI	End or Identify

Logic Levels

The interface and HP-IB bus use the standard +5V TTL logic level. The I/O backplane on the HP-85 uses a +6V logic level. A custom integrated circuit, called the Translator IC, has been developed to provide level shifting between these two levels. The Translator also provides buffering for information flow between the CPU in the HP-85 and the 8049 microcomputer on the interface. There are also four non-overlapping clock signals (Φ1, Φ2, Φ12, Φ21) appearing on the I/O backplane that have a +12V logic level. The interface uses two of these clock signals (Φ1 and Φ2).

The +5V TTL, +6V and +12V logic levels required for the low and high states are listed in Table 1-2.

Table 1-2. Logic Level Requirements

Signal States	HP-85 I/O Backplane Level Requirements	TTL Level Requirements	Clock Levels
Output High	⩾4V	⩾2.4V	⩽11.0V
Output Low	⩽.4V	⩽.4V	⩾0.4V
Input High	⩾3.6V	⩾2.0V	
Input Low	⩽.8V	⩽.8V	

Dimensions (Not including cable)	Approximately 16.7 x 12.7 x 1.5 cm (6.59 x 5 x .59 in)
Weight	0.5 kg (1.1 lb)

Line Drivers/Receivers

Each signal line connecting to the HP-IB bus has a driver and receiver circuit with the following characteristics:

Drivers		Receivers	
Type:	Open Collector		
Output Voltage Low State:	≤0.5V	Input Voltage Low State:	≤0.8V
Output Voltage High State:	≥2.5V (HP-IB Bus requires ≥2.4V)	Input Voltage High State:	≥2.0V

Cable Length

The length of the cable supplied with the interface is two metres. Also, see ''Cable Length Restrictions'' in Section 2.

Operating Temperature

0° to 55° C (32° to 131° F)

Power Requirements

The HP-85 mainframe supplies all power for the interface via the I/O backplane.

Data Transfer Rate

The HP-IB interface is capable of transferring data at a rate of 25k bytes per second in the Fast Handshake mode. Normal rates can be considerably slower and are determined by the program and the devices on the bus at a given time. The slowest device determines the data transfer rate.

Bus Functions

Interface functions are those elements which provide the capability for a bus device to send, receive and process messages if the device has the functional capability to do so.

Table 1-3 lists the HP-IB functions implemented by the 82937A interface. A complete description of these interface functions can be found in IEEE Standard 488-1978. Some of the terms used in the table are defined below.

Handshake	A technique used by devices to synchronize data byte transfers.
Source	An originator of data or commands.
Acceptor	A receiver of data or commands.
Listener	A device with Listen capability becomes Listener Active when it receives its Listen Address from the Controller Active device. As such, it is prepared to receive data bytes sourced by the Talker Active device.
Talker	A device with Talker capability becomes Talker Active when it receives its Talk Address from the Controller Active device. As such, it is prepared to source data bytes to one or more Listener Active devices. There can never be more than one device as a Talker at a given time.
System Controller	At power on, one (and only one) device on the HP-IB bus assumes the roll of System Controller. The HP-85 can be configured to be the System Controller by a switch on the interface circuit board. The System Controller resets the bus at power on and becomes Controller Active. Also, the System Controller may reset the bus and become Controller Active at any time, even if control has been passed to another bus device.
Controller Active	The Controller Active device configures the bus for the exchange of data by sourcing commands that designate one talker and one or more listeners. It can also send commands to cause specific actions to occur within a device, such as trigger, clear, etc. The Controller Active device can pass control to any other bus device capable of receiving control.
Serial Poll	The Controller Active device may serially poll another device to obtain it's status byte. This is usually done in user's software in response to a request for service. A device's status byte denotes the device's present status and whether or not it requested service.
Parallel Poll	The user can program the Controller Active device to conduct a parallel poll to obtain a status bit from devices on the HP-IB bus that are properly configured.

Table 1-3. Interface Functions and Allowed Capability

Mnemonic	Function/Allowed Capability
SH1	Source Handshake, complete capability. Provides the interface with the capability to send message bytes.
AH1	Acceptor Handshake, complete capability. Provides the interface with the capability to guarantee the proper reception of message bytes.
T6	Talker. Provides the interface with the capability to send device dependent data over the HP-IB bus to other devices. T6 implements: Basic Talker Serial Poll Unaddress if my listen address (MLA)
L4	Listener. Provides the interface with the capability to receive device dependent data from the HP-IB bus from other devices. L4 implements: Basic Listener Unaddress if my talk address (MTA)
SR1	Service Request, complete capability. Permits the interface to asynchronously request service from the Controller Active device.
RL1	Remote/Local, complete capability. Provides the interface with the capability to allow the HP-85 to select between either keyboard control (local) or program control (remote) if provided for in the user's program. The use of the RL function is totally program dependent.
PP2	Parallel Poll. Implemented by a jumper wire on the interface circuit board. Permits the interface to return a 1-bit status to the controller in response to a parallel poll.
DC1	Device Clear, complete capability. Gives the user access to the Device Clear state; use is program dependent.
DT1	Device Trigger, complete capability. Gives the user access to the Device Trigger state; use is program dependent.
C1,2,3,4,5	Controller. This implements: C1 – System Controller C2 – Send Interface Clear (IFC) and Take Charge C3 – Send Remote Enable (REN) C4 – Respond to Service Request (SRQ) C5 – Send Interface Message, Receive Control, Pass Control, Pass Control to Self, Parallel Poll, Take Control Synchronously

HP-IB Messages

The interface and I/O ROM determine how the preceding functions are implemented for passing messages over the bus. Following is a complete list of single line and multi-line messages. The manner in which some of these messages are responded to is device dependent and/or user defined. The controller is the primary source of messages that causes some specified action to take place on another bus device. Some of these messages are also discussed in more detail in Section 3.

Programming techniques for implementing these messages are not discussed in this manual. Refer to the I/O ROM manual for programming techniques.

Table 1-4. HP-IB Messages

Mnemonic	Message Name	Response
		Single Line Messages
ATN	Attention	The Controller Active places ATN true to source commands on the bus or in conjunction with EOI, to do a parallel poll. When ATN is false, data may be sent over the bus by a designated talker.
IFC	Interface Clear (Abort)	The System Controller uses this to place talkers and listeners in an unaddressed state. If control has been passed, the System Controller again becomes Controller Active when it sends IFC.
REN	Remote Enable	Removes all devices from Local Lockout mode and causes all devices to revert to manual control.
SRQ	Service Request	Indicates a device's need for interaction with the controller.
EOI	End or Identify	Terminates a flow of data, and can be used with ATN to do a parallel poll.
		Single Line Handshake Messages
DAV	Data Valid	Allows source to validate DIO lines.
NRFD	Not Ready For Data	Used to inform the source that all devices are ready for data.
NDAC	Not Data Accepted	Used by devices to inform the source that data has been accepted.

Table 1-4. HP-IB Messages

Mnemonic	Message Name	Response
		Multi-Line Messages
GTL	Go To Local	Causes selected devices to switch to local (front or rear panel) control.
LAG	Listen Address Group	A group of 31 listen addresses, one of which corresponds to the listen address of the interface.
UNL	Unlisten	Device becomes unaddressed to listen.
TAG	Talk Address Group	A group of 31 talk addresses.
UNT	Untalk	Causes a talker to become unaddressed to talk.
LLO	Local Lockout	Prevents local (front or rear panel) control of device functions.
DCL	Device Clear	Causes all devices to be initialized to a predefined or power-up state.
SDC	Selected Device Clear	Causes a device to be initialized to a predefined or power-up state.
SPD	Serial Poll Disable	Devices exit serial poll mode and are not allowed to send their status byte.
SPE	Serial Poll Enable	Devices enter serial poll mode and are allowed to send their status byte when addressed to talk.
GET	Group Execute Trigger	Signals one or more devices to simultaneously initiate a set of device dependent actions.
TCT	Take Control	Passes bus controller responsibilities from the current controller to a device which can assume the bus supervisory roll.
SCG	Secondary Command Group	A group of 32 commands which are only recognized if they immediately follow a talk or listen address.
DAB	Data Byte	Transfers device dependent information between a talker and one or more listeners. This may be programming information or data.

Note: Pressing the Reset key will reset the interface and return keyboard control to the operator. If the HP-85 is System Controller, pressing the Reset key will output the IFC message and set REN true.

Section 2

Installation

Unpacking and Inspection

If the shipping carton is damaged, ask the carrier's agent to be present when the interface is unpacked. If the interface is damaged or fails to meet electrical specifications, immediately notify the carrier and the nearest HP sales and service office. Retain the shipping carton for the carrier's inspection. The sales and service office will arrange for the repair or replacement of your interface without waiting for the claim against the carrier to be settled.

Installation

The select code switches, talk/listen address switches, system controller switch and the parallel poll jumper are preset at the factory as follows:

- Select Code 7
- Talk/Listen Address 21
- System Controller Enabled
- Parallel Poll Wired from E1 to E2 (DIO1)

Verifying or changing any of the above requires disassembly of the interface housing. If this is necessary, use the following disassembly procedure and disassemble the interface. Then refer to the discussion of the function requiring change.

If it isn't necessary to change or verify the factory settings, refer directly to ''Installing the Interface and Connecting Peripherals'' on page 20.

Disassembly

Refer to Figure 2-1 to see how the interface parts fit together. Place the interface on a flat surface with the side having the six screws facing upward and the cable coming out to the left. Then use the following steps to disassemble the interface.

1. Using a Pozidriv screwdriver, remove the six screws and set them aside.
2. Hold the interface parts together and turn the interface over so that the six screw holes are facing downward and the cable is still coming out to the left.
3. Hold the cable strain relief in place and remove the top half of the interface housing.

If you have followed the above steps, switch S1 should be oriented as shown in the following figures. If it isn't, orient it as illustrated before identifying the switch segments.

When you re-assemble the interface, reverse the above procedure, making sure the ground clip is in place. The ground clip should be under the circuit board when the component side is up.

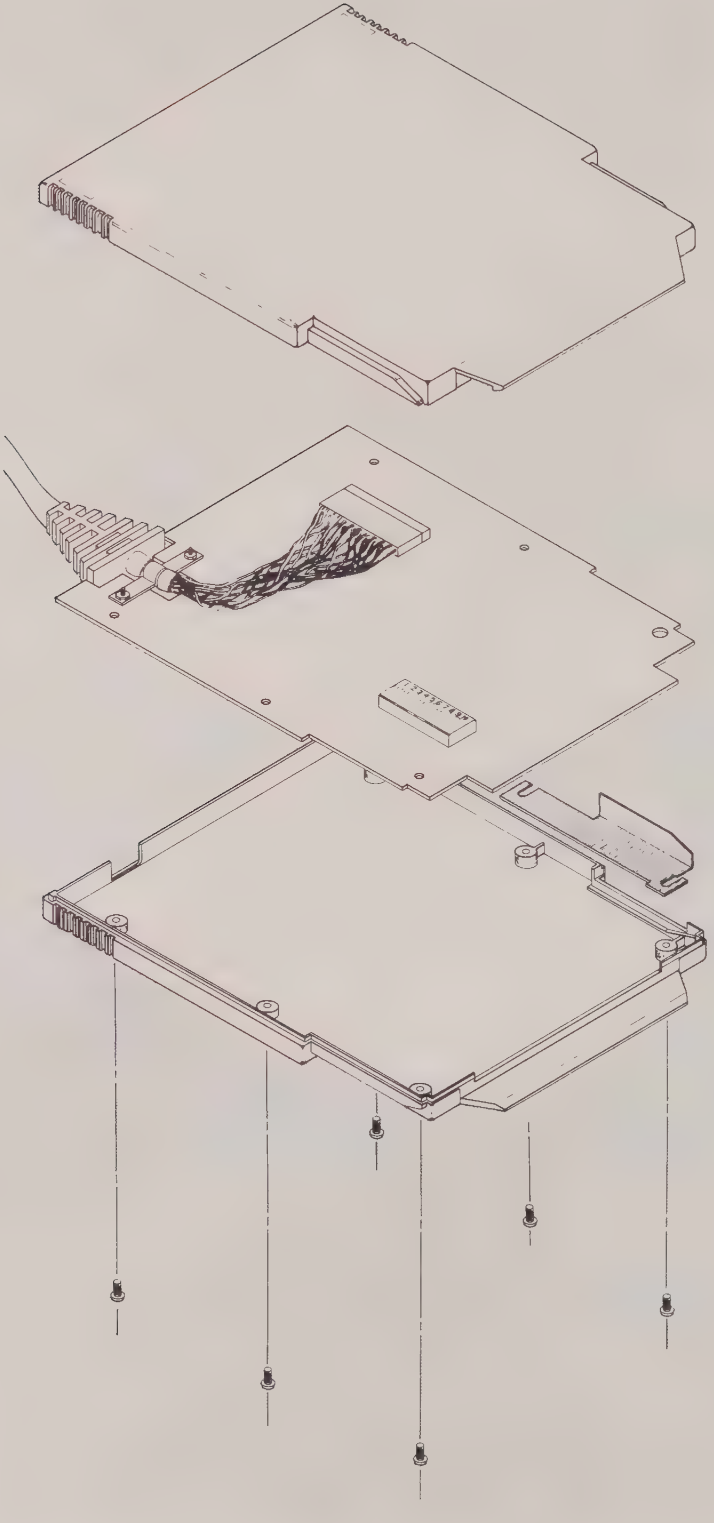


Figure 2-1. Disassembly

Switch Settings

As previously noted, the select code, talk/listen address and system controller functions are preset at the factory. Switch segments 2 through 10 of switch S1 on the interface circuit board are used for this purpose. Switch segment 1 is not used and may be in either position.

Once the interface is disassembled, any of the factory settings may be verified or changed by referring to the following discussions of the individual functions.

The interface may be equipped with either slide or rocker type switches. Both types are illustrated in the factory preset positions.

CAUTION

If you change any of the factory settings, make sure that you change the proper switch segments. Do not disturb the settings of adjoining switches. The small tip of a pencil or similar object is recommended for this purpose.

Select Code Switches

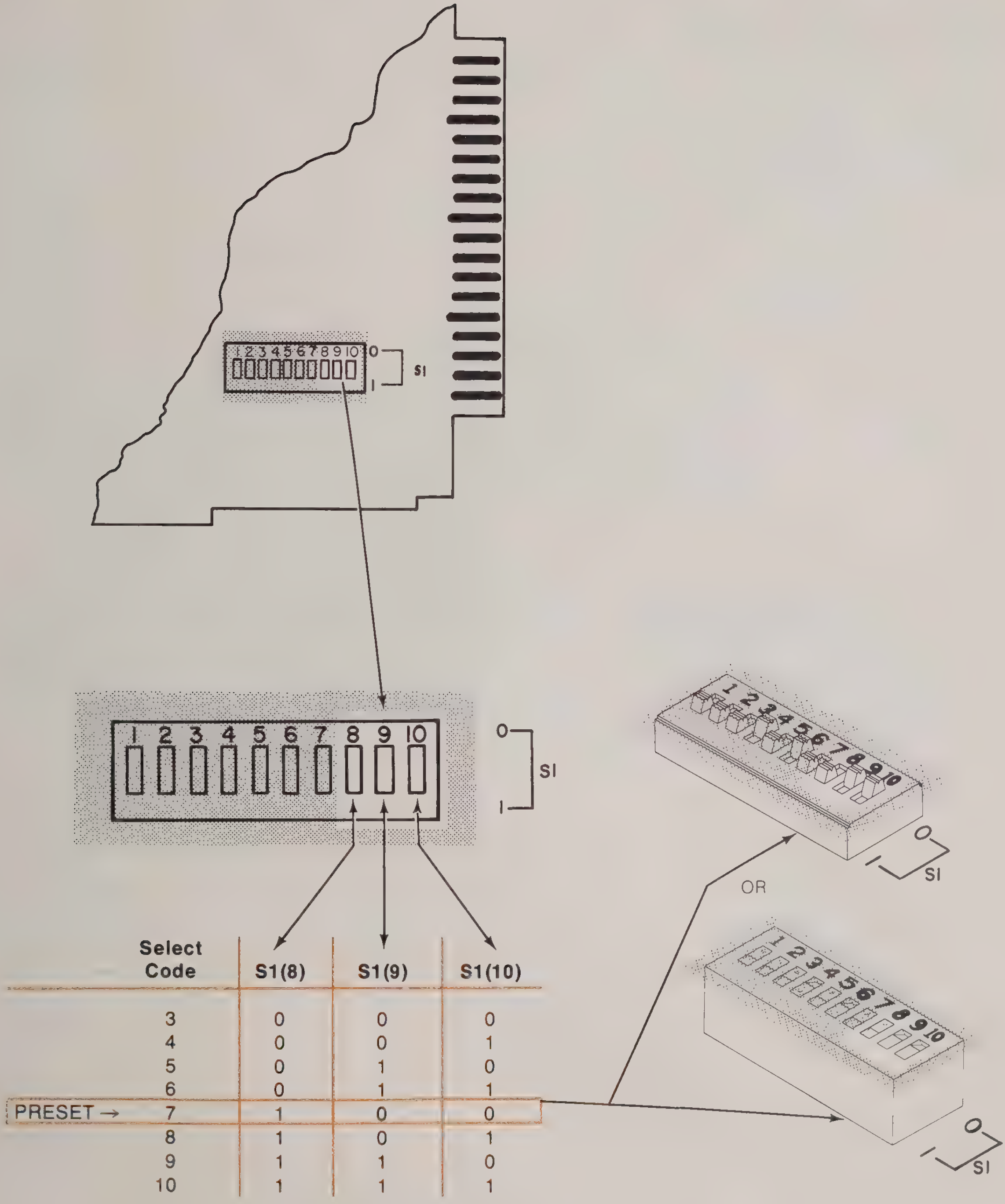
Switch segments 8, 9 and 10 of switch S1 are preset at the factory for select code "7" as follows:

- Switch segment 8 set to "1"
- Switch segment 9 set to "0"
- Switch segment 10 set to "0"

The "0" and "1" positions are labeled on the circuit board.

Select codes 3 through 10 may be set with these three switch segments. To change or verify the factory setting, orient the circuit board as shown in Figure 2-2 and locate switch segments 8, 9 and 10. **Next identify the "0" and "1" switch positions on the circuit board.** You may verify that select code "7" is properly set by comparing the actual positions of switch segments 8, 9 and 10 with those illustrated. They should be the same.

To change the select code, refer to the table and set switch segments 8, 9 and 10 as required for the select code chosen. For example, if select code "3" is to be set, the three switch segments must all be set to "0". In this case, if the switches are the slide type, slides 8, 9 and 10 must all be set to the "0" position; if they are the rocker type, all three rockers must be pressed down toward the "0" position.


1 2 3 4 5 6 7 8 9 10
0
1
S1
OR
PRESET →

Select Code	S1(8)	S1(9)	S1(10)
3	0	0	0
4	0	0	1
5	0	1	0
6	0	1	1
PRESET → 7	1	0	0
8	1	0	1
9	1	1	0
10	1	1	1

Figure 2-2. Select Code Switches

Note: Select codes 1 and 2 are the CRT and printer select codes, respectively.

Talk/Listen Address Switches

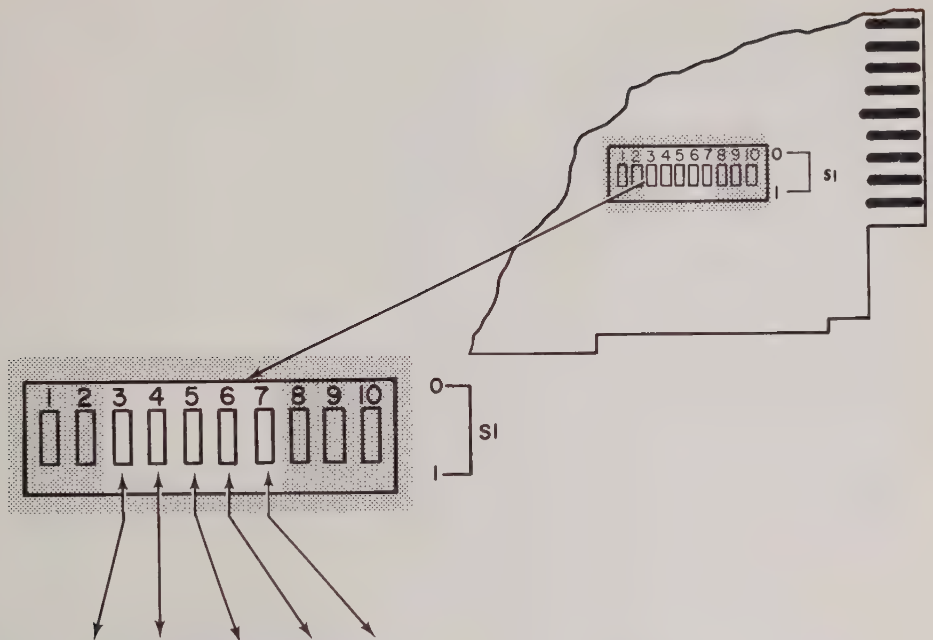
Switch segments 3 through 7 of switch S1 are preset at the factory for talk/listen address ''21'' as follows:

- Switch segment 3 set to ''1''
- Switch segment 4 set to ''0''
- Switch segment 5 set to ''1''
- Switch segment 6 set to ''0''
- Switch segment 7 set to ''1''

The ''0'' and ''1'' positions are labeled on the circuit board.

Any talk/listen address from 0 through 30 may be set with these five switch segments. To change or verify the factory setting, orient the circuit board as shown in Figure 2-3 and locate switch segments 3 through 7. **Next identify the ''0'' and ''1'' switch positions on the circuit board.** You may verify that talk/listen address ''21'' is properly set by comparing the actual positions of switch segments 3 through 7 with those illustrated. They should be the same.

To change the factory setting, refer to the table and set switch segments 3 through 7 as required for the talk/listen address chosen. For example, if talk/listen address ''30'' is chosen, switch segments 3, 4, 5 and 6 must be set to ''1'' and switch segment 7 set to ''0''. In this case, if the interface is equipped with slide switches, move the slides for segments 3, 4, 5 and 6 to the ''1'' position and the slide for segment 7 to the ''0'' position. If they are rocker switches, press the rockers down toward the ''1'' position for segments 3, 4, 5 and 6 and down toward the ''0'' position for segment 7.


1 2 3 4 5 6 7 8 9 10
0
1
SI
1 2 3 4 5 6 7 8 9 10
0
1
SI

	Address	S1(3)	S1(4)	S1(5)	S1(6)	S1(7)
	0	0	0	0	0	0
	1	0	0	0	0	1
	2	0	0	0	1	0
	3	0	0	0	1	1
	4	0	0	1	0	0
	5	0	0	1	0	1
	6	0	0	1	1	0
	7	0	0	1	1	1
	8	0	1	0	0	0
	9	0	1	0	0	1
	10	0	1	0	1	0
	11	0	1	0	1	1
	12	0	1	1	0	0
	13	0	1	1	0	1
	14	0	1	1	1	0
	15	0	1	1	1	1
	16	1	0	0	0	0
	17	1	0	0	0	1
	18	1	0	0	1	0
	19	1	0	0	1	1
	20	1	0	1	0	0
PRESET →	21	1	0	1	0	1
	22	1	0	1	1	0
	23	1	0	1	1	1
	24	1	1	0	0	0
	25	1	1	0	0	1
	26	1	1	0	1	0
	27	1	1	0	1	1
	28	1	1	1	0	0
	29	1	1	1	0	1
	30	1	1	1	1	0

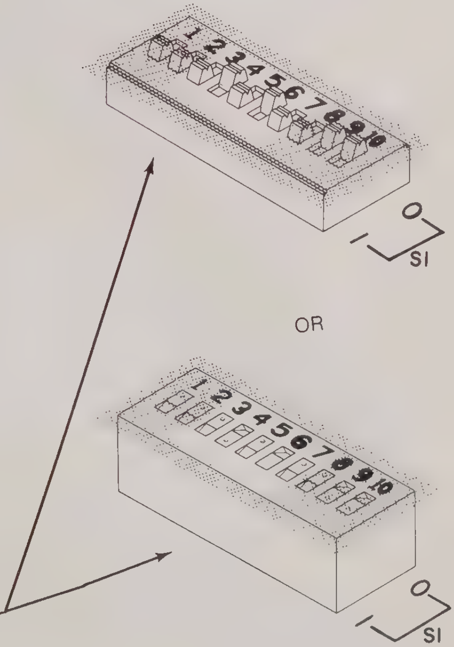

1 2 3 4 5 6 7 8 9 10
0
1
SI
OR
1 2 3 4 5 6 7 8 9 10
0
1
SI

Figure 2-3. Talk/Listen Address Switches

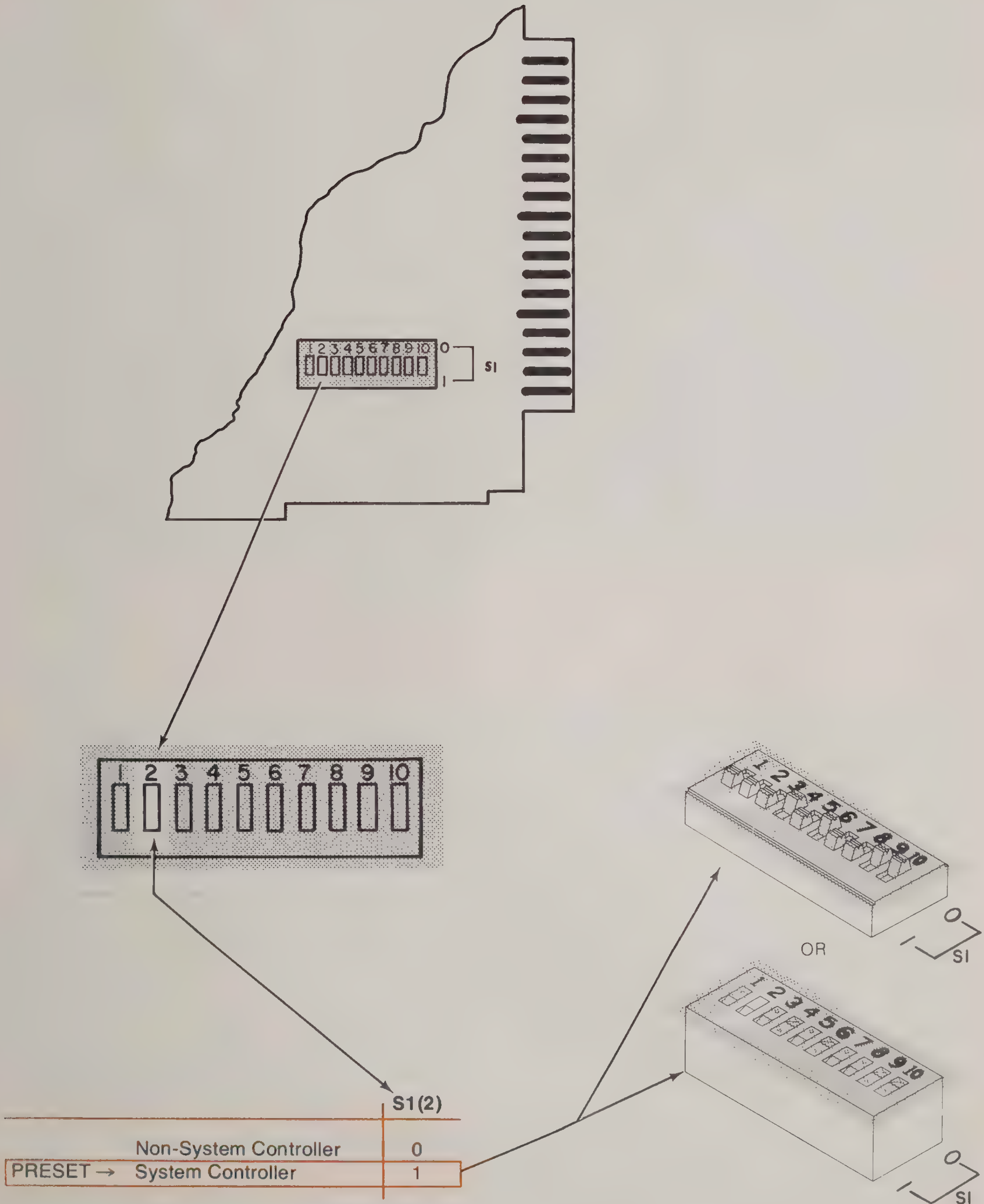
System Controller Switch

Switch segment 2 of switch S1 is preset to ''1'' at the factory to enable the host HP-85 as system controller. One (and only one) HP-IB bus device can assume system controllers status.

If the host HP-85 is to be the **system controller,** leave switch segment 2 in the factory preset position. To verify the factory setting, orient the circuit board as shown in Figure 2-4 and locate switch segment 2. **Next identify the ''0'' and ''1'' switch positions on the circuit board.** Switch segment 2 should be in the ''1'' position as illustrated.

If the host HP-85 is not to be the system controller, set switch segment 2 to the ''0'' position. In this case, if the interface is equipped with slide switches, move slide segment 2 to the ''0'' position. If they are rocker switches, press rocker segment 2 down toward the ''0'' position.

It is important that one, and only one, bus device be assigned as system controller. Failure to comply with this will result in improper bus operation.


1 2 3 4 5 6 7 8 9 10
0
1
S1
OR

	S1(2)
Non-System Controller	0
PRESET → System Controller	1

Figure 2-4. System Controller Switch

A jumper wire on the interface circuit board allows the interface to respond to a parallel poll by configuring the interface to one of the DIO lines. The interface is delivered with the jumper installed to assign DIO1 as the parallel poll response line.

The jumper is not necessary when the host HP-85 is the controller directing parallel polls to other HP-IB bus devices. It is necessary when the host HP-85 is being interrogated by another controller for a parallel poll response. In this case, the interface responds to the poll by pulling the DIO line it is configured to if it is requesting service.

The jumper is located underneath the cable wires. If it is necessary to change the DIO response line, first refer to Figure 2-5 and remove the cable connector. Remove the cable connector by prying it away from the circuit board connector as shown; do not pull on the cable wires to remove the connector. To change the parallel poll response line, refer to Table 2-1, locate the DIO line you want to configure and identify the two holes the jumper must be configured between. Then unsolder and remove the jumper wire using a soldering iron with a 25 watt or less rating. Next refer to Figure 2-6 and identify the two holes that you want to configure the jumper to. The odd-numbered holes are labeled on the component side of the circuit board; the even-numbered holes are labeled on the circuit side. Use the jumper you removed and solder in place between the two appropriate holes using rosin core solder. When you are finished, check both sides of the circuit board to ensure there are no bridges between the circuit board traces and re-connect the cable connector.

Table 2-1. Parallel Poll Jumper Placement

	DIO Line	Place Jumper From	To
PRESET →	DIO1	E1	E2
	DIO2	E3	E4
	DIO3	E5	E6
	DIO4	E7	E8
	DIO5	E9	E10
	DIO6	E11	E12
	DIO7	E13	E14
	DIO8	E15	E16

After all the switches are set and the parallel poll jumper is installed to meet your needs, re-assemble the interface before attempting to connect it to the HP-85.

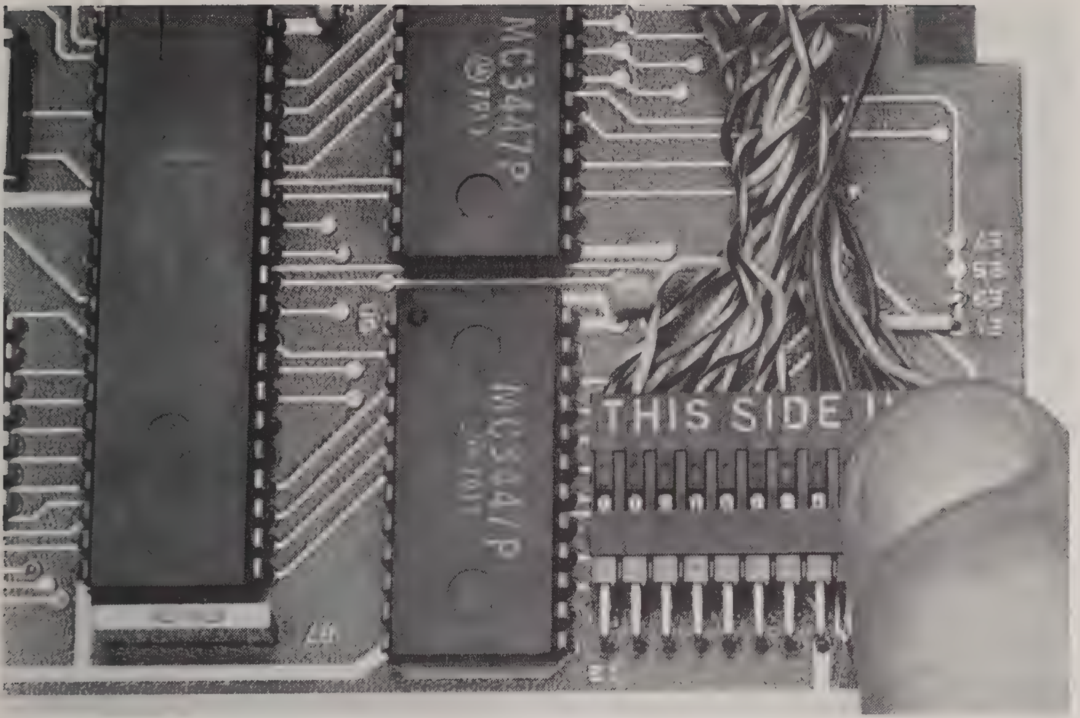

MC3447P
MC3447P
THIS SIDE U

Figure 2-5. Removing Cable Connector

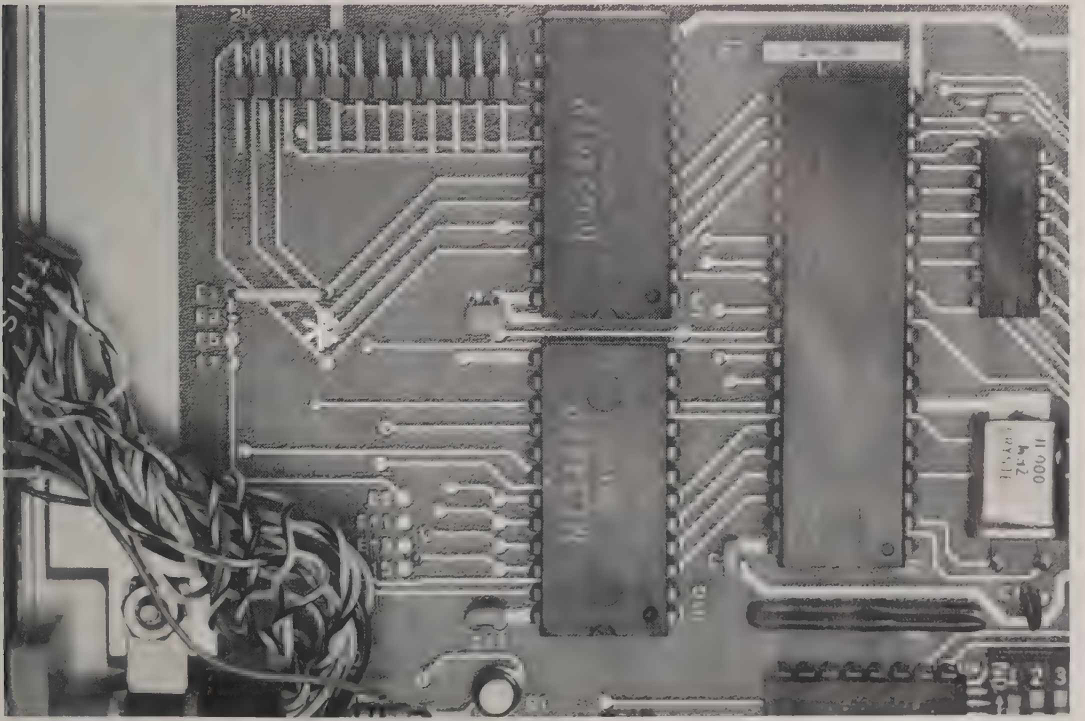


Figure 2-6. Parallel Poll Jumper Wire Placement

Installing the Interface and Connecting Peripherals

Make sure you read and understand this entire procedure before you install the interface or connect a peripheral device to it.

Safety Precautions

Manufacturers of peripheral devices do not all use the same grounding technique. Often, earth ground and logic ground are at different voltage levels. In some instances, this is deliberate in an effort to reduce ground return interference with digital signals.

When the 82937A interface is installed on the HP-85, earth ground and logic ground become connected together. Thus, if logic ground on a peripheral is never connected to earth ground or, if it is defective, it may have a voltage level considerably different than logic ground on the interface. This potential may be high enough to be hazardous unless peripherals are connected to the bus in an exacting manner.

If you don't know the grounding technique used on a peripheral, check with the manufacturer of the device. After verifying that suitable grounding techniques have been used in your peripheral, use the following steps in the order given to install the interface and peripherals to the HP-IB bus.

WARNING

To avoid personal injury and equipment damage, read and understand the preceding safety precautions and do not deviate from the order of the following steps when installing the interface and peripheral(s).

1. Turn the power switch located on the back of the HP-85 to the OFF position. However, make sure the HP-85 power cord is plugged into a grounded (three wire) AC outlet.
2. Remove the protective cover from any unused I/O slot to accommodate the interface and ROM drawer. Any unused I/O slots should be kept covered.
3. Refer to Figure 2-7 and install the interface into one of the I/O slots. Do not force the interface into the I/O slot. Notice that the tracks are keyed to prevent the interface from being inserted upside-down. If you intend to run the functional test, refer to that discussion at the end of this section before proceeding with step 4.

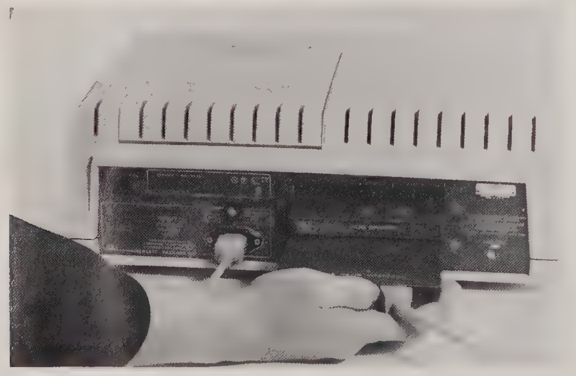


Figure 2-7. Installing the Interface

4. Make sure the power switch on all peripherals to be connected to the HP-IB bus are in the OFF position.
5. Connect the interface cable to the first peripheral. If other peripherals are to be connected at this time, first make sure they all have their power switches in the OFF position. Then, using the accessory cables listed on page 22 , install the cable connectors to the remaining peripherals in a piggy-back fashion, making sure to observe the cable length restrictions given on page 23 .
6. After all the peripherals are connected to the HP-IB bus, turn the power switches on the HP-85 and all peripherals to be operational on the bus to the ON position. A peripheral may be connected to the bus and have its power off without affecting bus operations, as long as more than 50% of the bus devices have their power on. For example, if there are three peripheral devices, two must have their power on; if there are two peripherals, both must have their power on.

Removing Peripherals/Disconnecting the Interface

Use the following steps in the order given to remove peripherals from the HP-IB bus or to disconnect the interface from the HP-85.

CAUTION

Do not remove the interface from the HP-85 with the HP-85 power switch on. Doing this will cause damage to either the interface, HP-85 or both.

1. Turn the power switch of all peripherals connected to the HP-IB bus to the OFF position.
2. Disconnect the HP-IB bus cable from each peripheral you intend to remove from the bus. If all peripherals aren't to be removed from the bus, you may return power to the remaining peripherals after the desired peripherals have been disconnected. If you intend to disconnect the interface from the HP-85, first make sure that all bus devices have their power switches in the OFF position. Then remove the interface cable from the peripheral it is connected to and proceed with step 3.
3. Turn the power switch on the back of the HP-85 to the OFF position. Make sure the interface cable is not connected to a peripheral and proceed with step 4.
4. Remove the interface from the I/O slot.

Interconnecting Cables

WARNING

Before you attempt to use any of the accessory cables, make sure you read and understand the safety precautions on page 20 .

The length of the interface cable is two metres. The end of the cable has a piggy-back connector that connects to a peripheral device. On some peripherals a physical interference problem may exist for the cable connector. Any peripheral that will not directly accept the cable connector will require the adapter. Other peripheral devices may be added to the bus by stacking the piggy-back connectors of the standard bus cables listed below.

Length	Accessory Number
½ metre	10833D
1 metre	10833A
2 metres	10833B
4 metres	10833C
Adapter	10834A

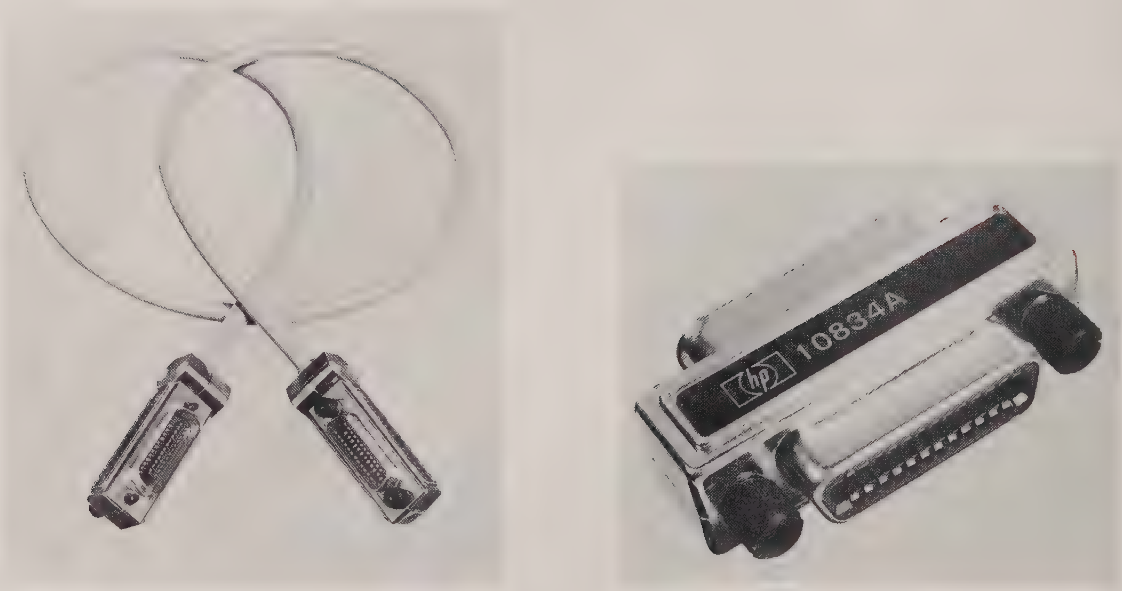


Figure 2-8. Bus Cables

There are no restrictions as to how cables may be connected together. However, it is recommended that no more than three or four piggy-back connectors be stacked together on one device. Doing so could exert enough force on the connector mounting to cause mechanical damage.

Cabling Length Restrictions

In order to ensure proper operation of the bus, two rules must be observed regarding the total length of bus cables when they are connected together:

- The total length of cable permitted with one bus interface must be less than or equal to two metres times the number of devices connected together with their power on (the interface is counted as one device).
- The total length of cable must not exceed 20 metres.

For example, there may be up to 4 metres of cable between the first two devices (2 devices x 2m/device = 4m). Additional devices may be added using 2-metre cables up to a total of 10 devices (10 device x 2m/device = 20 metres) using one 4-metre and eight 2-metre cables (4 + (8 x 2) = 20). If more than ten devices are to be connected together, cables shorter than two metres must be used between some of the devices. For example, 15 devices can be connected together using one 4-metre and thirteen 1-metre cables (4 + (13 x 1) = 17). Other combinations may be used as long as both requirements are met.

Metric Conversion Kit

The interface cable is supplied with mounting fasteners having metric threads. Other HP-IB instruments, however, may have either National Coarse (American) threads or metric threads. The American-threaded fasteners are chromium plated, while the metric-threaded fasteners are black.

Since metric and American threads cannot be connected together, a conversion kit may be needed. This kit will replace the mounting fasteners on any bus cable connector. Conversion kits can be ordered by specifying HP Part Number 5060-0138.

Functional Test

The following program listing may be entered and run to functionally test the interface. In most instances, running this test will indicate whether or not the interface is operational.

> **Note:** Before you run the test, make sure the interface has the System Controller switch segment set to "1" and the I/O ROM is installed.

Use the following steps to enter and run the test from the HP-85 keyboard:

1. Make sure the interface and I/O ROM are installed. If necessary, refer to page 20 and install the interface and I/O ROM.
2. Peripherals may or may not be connected to the interface.
3. Turn the HP-85 power switch to the ON position.
4. Press the SHIFT key and, while this key is depressed, press the AUTO key. This will cause the program lines to be numbered by ten in ascending order (10, 20, 30, etc.). Then press END LINE.
5. Refer to the following program listing and enter the first line by pressing these keys:

   C L E A R then press END LINE

   Continue this procedure until the entire program is entered, making sure to press END LINE as each numbered program line is entered.

```
 10 CLEAR
 20 DISP "82937A EXERCISER";"ENT
    ER SELECT CODE..." @ PRINT "
    82937A EXERCISER"
 30 INPUT S
 40 DISP "ENTER # OF TIMES TO RU
    N TEST..."
 50 INPUT N
 60 RESET S
 70 GOSUB 130
 80 GOSUB 370
 90 N=N-1
100 IF N>0 THEN GOTO 70
110 DISP "TEST COMPLETE" @ PRINT
     "TEST COMPLETE"
120 END
130 DISP "CHECKING SWITCH SETTIN
    GS AND";"HANDSHAKE"
140 A2=0
150 A1=53
[illegible]
[illegible]
[illegible]
190 A2=A2+1 @ IF A2>2 THEN GOSUB
     460
200 IF A2<3 THEN 220
210 GOTO 180
[illegible]
[illegible]
[illegible]
250 A2=A2+1 @ IF A2>4 THEN GOSUB
     460
```

```
260 IF A2=4 THEN 280
270 GOTO 240
280 WIO S,0;2
290 WIO S,1;120
300 WIO S,0;0
310 IF RIO (S,0)<1 THEN 310
320 A1=RIO (S,1)
330 IF RIO (S,0)#0 THEN GOSUB
    480
340 IF A1>192 THEN A1=A1-192
350 IF A1#53 THEN GOSUB 500
360 RETURN
370 DISP "STATUS TEST"
380 STATUS S,0 ; B1,B2,B3,B4,B5,
    B6
390 IF B1#1 THEN GOSUB 540
400 IF B2#0 THEN GOSUB 540
410 IF B3#64 THEN GOSUB 540
420 IF B4#0 THEN GOSUB 540
430 IF B5#A1 THEN GOSUB 540
440 IF B6#160 THEN GOSUB 540
450 RETURN
460 PRINT "HANDSHAKE ERROR-PROBA
    BLE";"PROCESSOR OR XLATOR FA
    ILURE"
470 RETURN
480 PRINT "XLATOR IB NOT CLEAR W
    HEN";"EXPECTED-PROBABLE XLAT
    OR OR";"PROCESSOR FAILURE"


490 RETURN
500 PRINT "SWITCHES DIDN'T READ
    AS EXPECTED DEFAULT"
510 A$=DTB$(A1)
520 PRINT "SWITCHES 2-7 READ AS
    "&A$[11,16]&"      EXPECTED
    110101"
530 RETURN
540 PRINT "STATUS ERROR"
550 PRINT "STS BYTES=";B1;B2;B3;
    B4;B5;B6
560 RETURN
```

6. After the program is entered press RUN. The printer should print:

```
                    82937A EXERCISER
```

   The screen should display:

```
82937A EXERCISER
ENTER SELECT CODE
?
```

7. Respond by entering the select code the interface is set to. For example, if select code 7 is set, press 7 and then END LINE. If the wrong select code is entered, one or more error messages will appear on the screen after step 8 is performed. If this happens, press RUN and enter the select code again.

8. The screen should then display:

```
ENTER # OF TIMES TO RUN TEST?
.
```

9. Enter the number of times you wish to run the short test. To run the test once, press 1 and then END LINE; to run the test ten times, press 1 0 and then END LINE, etc. Running the test several times may be useful if you suspect an intermittent failure.

10. The screen should then display:

```
CHECKING SWITCH SETTINGS AND
HANDSHAKE
```

If this test passes, shortly thereafter the screen will display:

```
STATUS TEST
```

The test names are displayed on the screen the number of times you have specified for the test to run unless there is a failure. If a test fails, an error message is printed for that test. If the first test fails, the printer should print:

```
HANDSHAKE ERROR-PROBABLE
XLATOR OR PROCESSOR FAILURE
```

OR

```
XLATOR IB NOT CLEAR WHEN
EXPECTED-PROBABLE XLATOR OR
PROCESSOR FAILURE
```

If the factory settings of switch segments 2 through 7 have been changed, you can always expect the printer to print:

```
SWITCHES DIDN'T READ AS EXPECTED
DEFAULT

SWITCHES 2-7 READ AS
```

(This portion of the printout will be a series of 0's and/or 1's depending on how the HP-85 interprets the switch settings. The first 0 or 1 is for switch segment 2, the second is for switch segment 3, etc. Switch segment 1 and 8 through 10 are not read.)

```
-EXPECTED 110101
```

(These are the factory settings.)

If the above is printed, it does not necessarily mean that the switches are being read improperly. Except for switch segment 2, which must always be 1 to enable the HP-85 as System Controller, the switches may be set to any combination. This feature permits you to set the switches for any talk/listen address given in Figure 2-3 and then verify the switches are being read properly for that address.

If the status test fails, the printer will print:

```
STATUS ERROR
STS BYTES =
```

(This portion of the printout will be a series of numbers indicating which bits are set in status registers SR0 through SR5. SR6 is not read. The first number is for SR0, the second for SR1, etc. These status registers are shown on page 42 .)

An example of a proper status byte would be:

```
1   0   64   0   53   160
```

The only number that may vary in a proper status byte is the one indicating the bits set in SR4. This is shown as 53 in the above status byte and is discussed in more detail in the following discussion.

If a status error is printed, you can identify which bits of a register are set by decoding the decimal number printed into its binary equivalent. Then refer to the status registers on page 42 . The status register bits are shown with bit 0 being the least significant and bit 7 the most significant. The decimal values of the bits are as follows:

SR Bit	Decimal Equivalent
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

The numbers of the proper status byte can therefore be interpreted as follows:

- The first number, 1, indicates that bit 0 in SR0 is set to 1 and all others are 0.
- The second number, 0, indicates that all bits of SR1 are 0.
- The third number, 64, indicates that bit 6 of SR2 is set to 1 and all others are 0.
- The fourth number, 0, indicates that all bits of SR3 are 0.
- The fifth number, 53, indicates that bits 5, 4, 2 and 1 of SR4 are set to 1 and all others are 0.
- The sixth number, 160, indicates that bits 7 and 5 of SR5 are set to 1 and all others are 0.

Now, let us examine the fifth number, 53, which is the only number that may vary in a proper status byte. This number reflects the contents of SR4, the address register. This register should always coincide with how switch segments 2 through 7 are set, which was discussed earlier in this section. The number 53 reflects that the system controller switch and the talk/listen address switches are all in the factory preset positions. The smallest number that should ever appear in this position while running the test is 32, corresponding to the system controller switch being set to 1 and the talk/listen address switches set for address 0. The largest number that should ever appear in the fifth position is 62, corresponding to the system controller switch being set to 1 and the talk/listen address switches set for address 30.

11. After all the tests are run, the printer and screen should display:

```
TEST COMPLETE
```

Section 3

Theory of Operation

Introduction

This section contains a description of the interface circuit operations. This includes a detailed block diagram description of the Translator IC and a schematic diagram description of various interface components and signal lines. Although this information is provided, component level repair is not recommended due to the microcomputer based organization of the interface.

If the interface appears to be malfunctioning, it is recommended that you contact your nearest HP sales and service office for assistance; office locations are listed after the schematic diagram.

Translator IC Description

The interface uses an 8049 microcomputer (μC) which requires the +5V TTL logic level. The microprocessor in the HP-85 uses a +6V logic level. A special IC, known as a Translator, permits communication between the two devices by providing level translation. The +5V and +6V power supplies and two +12V clock signals ($\Phi 1$ and $\Phi 2$) required by the Translator are located on the HP-85 mainframe. They are brought out to the interface via the I/O backplane when the interface is inserted into one of the four I/O slots.

The Translator supports the following operations:

- Handshaking of data and command information from the HP-85 to the μC.
- Handshaking of data from the μC to the HP-85.
- Interrupts issued to the HP-85 by the μC.
- Interrupts issued to the μC by the HP-85.
- Fast handshake operation where the Translator halts the HP-85 with each data byte transfer to synchronize the flow of data.

It may be helpful to refer to the block diagram of the Translator IC (Figure 3-1) and the interface schematic diagram (Figure 3-9) while reading the theory discussed in this section.

Select Codes

Interfaces are memory mapped. When the HP-85 addresses an interface, it provides an address over the address and data bus (B0X – B7X). The Translator demultiplexes the address and then compares address bits (A1, A2, A3) with select code bits (SC0, SC1, SC2). If they match, the interface is addressed. When addressed, another bit (A0) is sent out by the Translator's Address Demultiplexer. If A0 is low, the HP-85 writes to the control register (CR) and reads the status register (SR). When A0 is high, the HP-85 writes to the output buffer (OB) and reads the input buffer (IB). These four registers are discussed next.

Translator I/O Registers

The HP-85 sends data to the μC via the output buffer register (OB) and defines that data by setting appropriate bits in the control register (CR). Both registers are physically located within the Translator IC. The OB is write-only by the HP-85 and read-only by the μC. Except for bit 0 and bit 7, the CR is also write-only by the HP-85 and read-only by the μC. Different status bits are read by the μC for bit 0 and bit 7 of the CR than those written as bits 0 and 7 by the HP-85. This is illustrated in Figure 3-2.

HP-85 Write to CR

7	6	5	4	3	2	1	0
RESET	0	0	0	0	CED	COM	CINT

μC Read CR

7	6	5	4	3	2	1	0
OBF	0	0	0	0	CED	COM	IBF

Figure 3-2. CR Bit Assignments

CINT	Interrupt. This bit is routed to the T1 input of the μC which is used to interrupt.
COM	Command.
CED	End Data. Used by HP-85 to terminate a data transfer to the μC.
RESET	Resets the μC. This bit is routed to the reset input of the μC.
IBF	Input Buffer Full.
OBF	Output Buffer Full.

Two other registers within the Translator are used when the HP-85 receives data from the μC. These registers are the input buffer register (IB) used to handle the data and the status register (SR) which contains status bits to implement communication protocol. The IB is write-only by the μC and read-only by the HP-85. Except for bit 0 and bit 7, the SR is also write-only by the μC and read-only by HP-85. Different status bits are read for bits 0 and 7 by the HP-85 than those written as bits 0 and 7 by the μC. This is illustrated in Figure 3-3.

μC Write to SR

7	6	5	4	3	2	1	0
HLTEN	0	0	0	PACK	PED	BUSY	SERVICE

HP-85 Read SR

7	6	5	4	3	2	1	0
OBF	0	0	0	PACK	PED	BUSY	IBF

Figure 3-3. SR Bit Assignments

SERVICE	The μC sets this bit to initiate an interrupt to the HP-85.
BUSY	Informs the HP-85 the state of the μC. When low, the μC is monitoring OBF to determine when the next command or data byte is available from the HP-85. When OBF is set, the μC sets BUSY to 1, reads the OB and CR, performs the necessary operation then returns BUSY to low.

PED	Processor (μC) End Data. Set high by the μC upon detection of a programmed termination sequence (byte count, end character or EOI).
PACK	Processor (μC) Interrupt Acknowledge. The μC's interrupt service routine sets this bit after being interrupted by the HP-85.
HLTEN	Used by the μC to halt the HP-85 based on the status of OBF and IBF during fast handshake data transfers.
IBF	Input Buffer Full.
OBF	Output Buffer Full.

Interrupts

The interface can interrupt the HP-85 for several reasons. For example, an interrupt may be generated when the interface detects the SRQ line being asserted from the HP-IB bus if the user has programmed the SRQ bit in the interrupt mask register. The interface interrupts the HP-85 as follows:

1. The interface μC sets bit 0 (Service) true in the Translator status register. This causes the interrupt line, $\overline{\text{IRLX}}$, to go low, interrupting the HP-85.
2. The HP-85 acknowledges the interrupt and obtains from the interface an interrupt vector. This tells the HP-85 that this is an I/O interrupt as opposed to an interrupt from the internal keyboard chip.
3. After the interrupt is acknowledged, $\overline{\text{IRLX}}$ is returned high and all other interfaces are locked out from interrupting until re-enabled by the HP-85.
4. The HP-85 then obtains the select code of the interrupting interface and processes the interrupt accordingly.
5. After the interrupting interface is serviced, the other interfaces are re-enabled to interrupt.

Since the HP-85 can accommodate up to three interfaces, there must be a priority scheme if more than one requests service at the same time. This would most likely be a very rare occurrence. Essentially, the HP-85 handles interrupts on a first-come, first-served basis. However, if more than one interface is pulling the $\overline{\text{IRLX}}$ line before the HP-85 acknowledges the interrupt request, the interface that gets serviced first will be the one occupying the upper-most I/O slot.

Refer to the flow chart of Figure 3-4 as you read the following sequence that takes place when the HP-85 interrupts the μC on the interface.

The μC interrupt counter is preset to 255. The interrupt is initiated when the HP-85 pulls $\overline{\text{CINT}}$ of the Translator (T1 of the μC) low. This increments the μC interrupt counter from 255 to 0 (an overflow), and starts the interrupt sequence. Before the μC acknowledges the interrupt, it reads the CR to check the IBF bit. If the IBF bit is set, it indicates the IB contains data. To prevent this data from being lost, the μC sets an input buffer restore flag. This will permit the IB to be restored after the interrupt is serviced.

The μC acknowledges the interrupt by setting the PACK bit in the SR. When PACK is set, the HP-85 reads the IB, which may or may not contain data, and discards it. The HP-85 then sets the COM (command) bit in the CR, writes an instruction into OB and returns $\overline{\text{CINT}}$ high. When $\overline{\text{CINT}}$ goes high, the μC accepts the instruction and performs the operation specified.

After the operation is performed, the HP-85 strobes $\overline{\text{CINT}}$ again, which increments the μC interrupt counter from 0 to 1. When this occurs, the interrupt counter gets reset to 255 and the SR is restored. If the input buffer restore flag is set, the IB is also restored. The μC then returns from interrupt.

Notice in the above sequence that $\overline{\text{CINT}}$ was strobed twice. The first time caused an overflow of the μC interrupt counter (from 255 to 0) which started the interrupt sequence. The second strobe incremented the interrupt counter from 0 to 1 which caused the μC to return from interrupt.

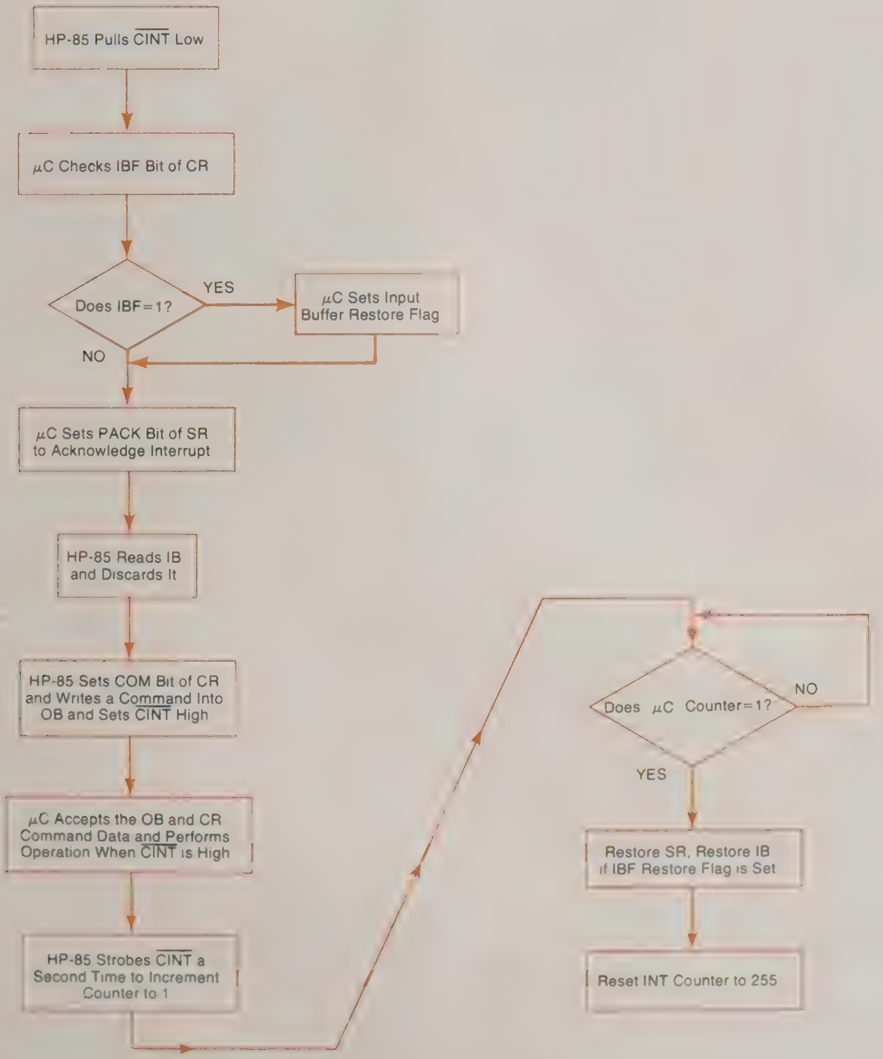

HP-85 Pulls $\overline{\text{CINT}}$ Low
μC Checks IBF Bit of CR
Does IBF=1?
YES
μC Sets Input Buffer Restore Flag
NO
μC Sets PACK Bit of SR to Acknowledge Interrupt
HP-85 Reads IB and Discards It
HP-85 Sets COM Bit of CR and Writes a Command Into OB and Sets $\overline{\text{CINT}}$ High
μC Accepts the OB and CR Command Data and Performs Operation When $\overline{\text{CINT}}$ is High
HP-85 Strobes $\overline{\text{CINT}}$ a Second Time to Increment Counter to 1
Does μC Counter=1?
NO
YES
Restore SR, Restore IB if IBF Restore Flag is Set
Reset INT Counter to 255

Figure 3-4. Interrupt Flow Chart

HP-85 I/O Backplane Lines

The I/O lines in Table 3-1 appear at each of the four I/O slots on the HP-85. They connect to the interface when it is inserted into any of the slots. Figure 3-5 shows how the I/O slot connector pins are numbered. Each I/O slot is numbered in the same manner.

Table 3-1. HP-85 I/O Backplane Lines

Line	Pin No.	Meaning	Direction HP-85 Interface
Φ21	1	Non-Overlapping Clock (+12V) (not used)	→
Φ2	2	Non-Overlapping Clock (+12V)	→
Φ1	3	Non-Overlapping Clock (+12V)	→
Φ12	4	Non-Overlapping Clock (+12V) (not used)	→
V_{GG}	5	+12V	→
$\overline{\text{LMAX}}$	6	Load Memory Address	→
$\overline{\text{IRLX}}$	7	Interrupt Request	←
$\overline{\text{RDX}}$	8	Read	→
$\overline{\text{RC}}$	9	Read Control	←
$\overline{\text{WRX}}$	10	Write	→
V_{CC}	11	+5V	→
B4X	12	I/O Line	↔
B5X	13	I/O Line	↔
B6X	14	I/O Line	↔
B7X	15	I/O Line	↔
—	16	Open	
$\overline{\text{PRIH}}$	17	Priority High	→
DIS	18	Disable. +6V Signal Voltage. This is used to keep $\overline{\text{PRIH}}$ and $\overline{\text{PRIL}}$ from shorting together when an interface is plugged into an I/O slot.	
SPKR	$\overline{1}$	Speaker Output (not used)	→
—	$\overline{2}$	Open	
—	$\overline{3}$	Open	
L. GND	$\overline{4}$	Logic Ground	→
L. GND	$\overline{5}$	Logic Ground	→
L. GND	$\overline{6}$	Logic Ground	→
$\overline{\text{HALTX}}$	$\overline{7}$	Halt	←
V_{BB}	$\overline{8}$	−5V	→
—	$\overline{9}$	−12V (not used)	→
PWO	$\overline{10}$	Power On	→
V_{CC}	$\overline{11}$	+5V	→
B0X	$\overline{12}$	I/O Line	↔
B1X	$\overline{13}$	I/O Line	↔
B2X	$\overline{14}$	I/O Line	↔
B3X	$\overline{15}$	I/O Line	↔
V_{DD}	$\overline{16}$	+6V	→
—	$\overline{17}$	Open	
$\overline{\text{PRIL}}$	$\overline{18}$	Priority Low	←

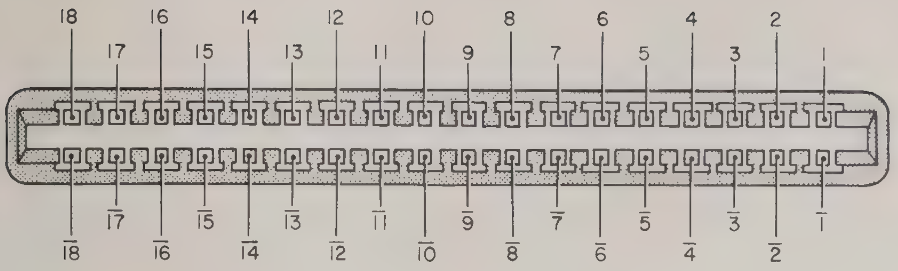

18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
$\overline{17}$
$\overline{15}$
$\overline{13}$
$\overline{11}$
$\overline{9}$
$\overline{7}$
$\overline{5}$
$\overline{3}$
$\overline{1}$
$\overline{18}$
$\overline{16}$
$\overline{14}$
$\overline{12}$
$\overline{10}$
$\overline{8}$
$\overline{6}$
$\overline{4}$
$\overline{2}$

Figure 3-5. HP-85 I/O Slot Connector Pinout

HP-IB Bus Lines

The standard HP-IB signal lines are described next. The function of each line is defined by IEEE Standard 488-1978.

Note: IEEE Standard 488-1978 specifies that the HP-IB bus use negative true logic which is often confusing or misunderstood. This explanation of negative true logic, as it applies to the HP-IB bus and interface, is given so that no misconceptions exist.

All HP-IB bus lines are shown on the schematic without bars over them. Thus, if a bus line is true, it will have a low voltage level. Positive true logic is used on the interface. In positive true logic a line whose mnemonic does not have a bar over it is true when it has a high voltage level. As an example, on the schematic bus line ATN is shown as $\overline{\text{IATN}}$ when it appears as an internal signal on the interface. Thus, the voltage level representing a true ATN on the HP-IB bus is the same as a true $\overline{\text{IATN}}$ on the interface.

Data Lines (DIO1 – DIO8)

The data lines are used to communicate all data including input, output, program codes, status and control information between instruments connected to the bus. One character byte is sent at a time in a byte-serial, bit-parallel fashion. In most instruments, characters are based on the 8-bit ASCII representation. For instance, if the interface is connected to a printer, the 8-bits that represent one character are all sent at once in parallel. Then, the next character is sent and so on. Thus, the 8-bits (one byte) defining a character are all sent at once in parallel (bit-parallel) while each character is sent serially (byte-serial).

Control Lines (ATN, IFC, SRQ, EOI and REN)

The five control lines govern and define the information on the DIO lines.

ATN (Attention) is driven by the active controller. When high, the information on DIO1 through DIO8 is interpreted as data. When low the information is control or address information.

IFC (Interface Clear) is used only by the system controller to initialize the bus. When IFC is held low for at least 100μs, any talker or listener is stopped, the serial poll mode is disabled and control is returned to the system controller, regardless of the device that had control of the bus. When IFC is high, it has no effect on bus operations.

SRQ (Service Request) is pulled low by a device when it wants the attention of the controller. SRQ may be driven low at any time except when IFC is low.

EOI (End or Identify) may be used to indicate the end of a string of characters. For example, if ATN is high to indicate the information on DIO1 through DIO8 is data, the talker may indicate the end of its data transmission by driving EOI low when it places the last byte on the data lines.

REN (Remote Enable) is driven by the system controller and is one of the lines necessary for an instrument to operate under remote control. Only instruments capable of remote operation use REN and they monitor it at all times. Instruments that do not use REN terminate the line into a resistor load. The system controller may change the state of REN at any time.

Transfer Lines (NRFD, NDAC and DAV)

Three transfer (handshake) lines are used to execute the transfer of each byte of information on the data lines. They allow asynchronous data transfer without timing restrictions being placed on any instrument connected to the bus. The transfer speed of each data byte is determined by the speed at which the slowest instrument is capable of sending or receiving data.

NRFD (Not Ready For Data) is high to indicate that all listeners are ready to accept information on the DIO lines. When NRFD is low, at least one listener is not ready for data.

NDAC (Not Data Accepted) is high to indicate that all listeners have accepted the information on the DIO lines. When NDAC is low, at least one listener has not accepted the data.

DAV (Data Valid) is low to indicate the information on the DIO lines is valid for the listener(s). When DAV is high, the information on the DIO lines is not valid.

Data Transfer

Transfer of data on the bus is asynchronous. That is, there are no restrictions placed on the data rates of instruments connected to the bus. The slowest instrument involved in a data transfer on the bus determines the rate of data transfer. As previously mentioned, transfer is under the control of three lines: DAV, NRFD and NDAC. The talker controls the data lines and DAV; the listener(s) controls NRFD and NDAC.

The transfer of a byte of data is initiated by all listeners setting NRFD high, indicating they are ready for data. When the talker recognizes NRFD is high it places data on the lines and validates the data by placing DAV low. The listener(s) senses that DAV is low, accepts the data, then notes the acceptance by setting NDAC high. Notice that the assertive, or action state, of NRFD and NDAC is high. Because all instruments on the bus have their corresponding lines connected together, all listeners must have NRFD and NDAC high before the respective line is high. This is because a low state overrides a high. Thus, a slow listener on the bus that does not accept data and thereby assert NRFD and NDAC as readily as other listeners will slow the rate of data transfer.

The timing diagram in Figure 3-6 shows the relationship of these signal lines during a data byte transfer. At t_1 NRFD is set high to indicate all listeners are ready to accept data. At t_2 the talker places data on the lines and indicates the validity of the data at t_3 by placing DAV low. At some interval between t_3 and t_4 a listener may place NRFD low but it must do so before, or at the same time that NDAC is set high. The talker may return DAV high after it detects NDAC is high, which is shown at t_5. A listener may set NDAC low (shown between t_5 and t_6) but, it must do so before NRFD is set high at t_6. When NRFD goes high at t_6, a new cycle begins equivalent to t_1.

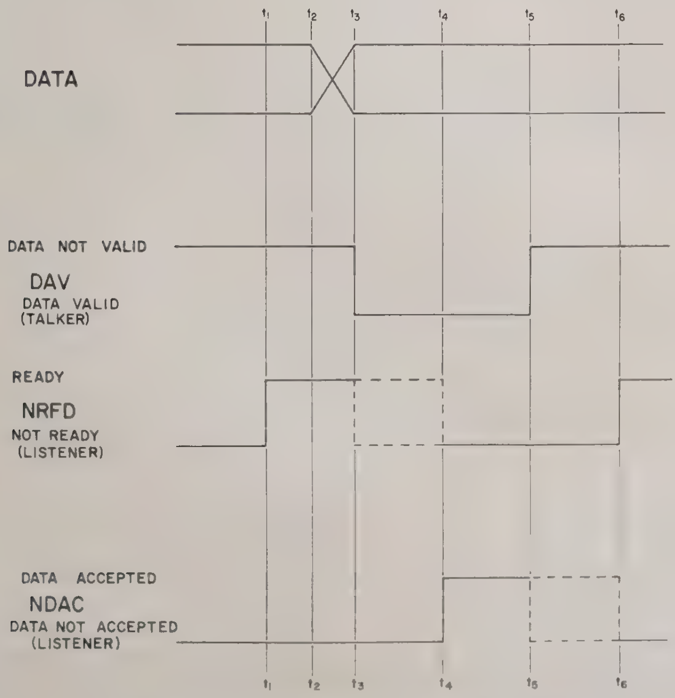


Figure 3-6. Data Transfer Timing Diagram

Polling

Polling is a special bus activity that permits a controller to determine the operating status of other devices. These two processes, serial poll and parallel poll, are an intricate part of the interrupt protocol. However, because software can define whether a serial or parallel poll (or both) is implemented, they are discussed as separate topics to simplify the explanation.

Serial Poll

A serial poll permits the controller active device to obtain a status byte from any bus device that supports the serial poll function. When the controller active device detects that a service request exists (SRQ line is pulled), it may serially poll the bus devices expected to have requested service. When a device is polled, it always returns its status byte. Bit 7 of the status byte will be set to 1 if the device requested service or it will be 0 if it did not. The remainder of the bits in the status byte can be used to indicate the reason for the service request and are totally device dependent.

The programmer has two means of detecting when SRQ is pulled:

- Perform a STATUS read of register SR2 to see if the SRQ bit (bit 5) is set;
- Enable an End-of-Line interrupt when SRQ is pulled.

The bus might have only one bus device capable of requesting service and perhaps for only one purpose. In this case, a serial poll may not be needed. However, if a single device can pull SRQ for various reasons or, if more than one bus device is capable of pulling SRQ, then a serial poll can be used to determine which device(s) requested service and why.

When the HP-85 is controller active and initiates a serial poll, the program specifies the select code and the address of the device to be polled. The 82937A interface then sends the following sequence over the bus:

1. ATN is set true.
2. UNL (Unlisten) command is sent to the device.
3. Interface sends its own listen address and the peripheral talk address commands.
4. SPE (Serial Poll Enable) is sent over the bus, followed by ATN going false. The device being polled then sends its status byte. If it was the only device requesting service, SRQ is removed. If it was not the only device pulling SRQ, or if it didn't request service, SRQ remains, indicating some other device requires service.
5. ATN is set true again followed by the SPD (Serial Poll Disable) command.
6. The UNT (Untalk) command is then sent over the bus causing the peripheral to cease being a talker.

Note in the above sequence that SRQ may or may not be removed when a device is polled. SRQ is removed if the device being polled is the one and only device requesting service. Therefore, the user's program must specify the order in which bus devices are polled and serviced.

If the HP-85 is not the controller active device, the user may allow it to request service with the REQUEST statement followed by a program dependent status byte. Bit 7 of this status byte maps directly to SRQ on the bus. The HP-85 provides this status byte when it is serially polled by the controller active device. If it was requesting service, bit 7 will be set to 1; otherwise, it will be 0. The HP-85 will remove SRQ when it sends its status byte if it was the one and only bus device requesting service. However, if SRQ is removed, bit 7 remains set until changed by the program.

Parallel Poll

In a parallel poll each bus device is assigned one of the DIO lines for identification purposes. This is configured within each bus device. The parallel poll jumper on the interface is used to assign one of the DIO lines for identification of the interface if it is to respond to parallel polls.

A parallel poll can be much faster than a serial poll because the controller reads DIO1 through DIO8 all at once to determine which device(s) requested service.

Switch Buffer (U4)

The switch buffer is a single integrated circuit that isolates switch lines A0 through A4 and SC from the interface address data bus. These lines were discussed in Section 2.

The outputs of U4 connect to the address-data bus and the external control latch (U3). Unless it is enabled, these outputs remain in a high impedance state. When it is enabled via inputs DIS1 and DIS2, the switch settings are reflected on the outputs and therefore appear on D0 through D5 of the address-data bus and the inputs to the external control latch. Thus, the switch buffer is enabled only during initialization or when the 8049 μC is reset to prevent it from interfering with other operations. Once the status of the switches is read by the μC it is not necessary to do so again unless the μC is reset.

To read the switch buffer, the μC sends an address to the Translator read/write logic via the address-data bus and latches it with the address latch enable (ALE) signal. The Translator responds by placing $\overline{\text{ADR3}}$ low. The μC then places $\overline{\text{RD}}$ low. During the period when both these signals are low, the switch buffer is enabled allowing the switch settings to be reflected on D0 through D5 of the address-data bus to be read by the μC.

External Control Latch (U3)

This latch is clocked to accept the information on the address-data bus lines (D0 – D5). This occurs when $\overline{\text{WR}}$ is low and the μC writes an address to the Translator that causes $\overline{\text{ADR3}}$ to go low. This latch is cleared on initialization or when the μC is reset.

The output lines of this latch serve a variety of functions and are discussed separately.

Output	**Function**
EN NDAC	The IEEE Standard 488-1978 specifies that when ATN goes true, NDAC must go false within 200 ns. This helps ensure that a transfer will proceed in an orderly fashion. The EN NDAC output is gated with IATN to provide a hardware implementation to meet this specification.

EN ATN INT	This line is gated with the IATN line to the interrupt ($\overline{\mathrm{INT}}$) input of the μC. When EN ATN INT is set and a peripheral device asserts ATN, the μC recognizes an interrupt request.
EN REN INT	This line is gated with the $\overline{\mathrm{IREN}}$ line to the interrupt ($\overline{\mathrm{INT}}$) input of the μC. When EN REN INT is true and a peripheral device asserts REN, the μC recognizes an interrupt request.
EN IFC INT	This is gated with the IIFC line to the interrupt ($\overline{\mathrm{INT}}$) input of the μC. When EN IFC INT is set and a peripheral device assert IFC, the μC recognizes an interrupt request.
TALKER ACTIVE (TA)	When TA and $\overline{\mathrm{IATN}}$ are both true it enables the bi-directional buffers connected to the HP-IB bus (DIO1 through DIO8) in a direction which permits the μC to output information to the bus.
CONTROLLER ACTIVE (CA)*	When this line and IATN are both set the μC is allowed to output information to the HP-IB bus in the same manner described above. Also, the controller can only source commands when ATN is true. The CA line is also used to enable the buffers for the service request (SRQ) and attention (ATN) lines on the HP-IB bus. In the latter, the controller can only source commands when ATN is true; while another talker can only source data when ATN is false.

Bi-directional Bus Transceivers (U9, U10)

Two integrated circuits provide the bi-directional bus transceiver circuitry used to transmit or receive information between the HP-85 and other HP-IB bus devices. The send/receive pairs that connect to bus lines DIO1 through DIO8 are enabled so that all either send or receive bus information in the same direction simultaneously. The handshake and control lines for the bus are enabled as follows:

NRFD and NDAC	Two send/receive pairs are enabled to send or receive the state of these bus lines in the same direction simultaneously.
DAV and EOI	Two send/receive pairs are enabled to send or receive the state of these bus lines in the same direction simultaneously.
ATN	One send/receive pair is enabled by the CA line to permit ATN to be asserted from the controller to other bus devices.
SRQ	One send/receive pair is enabled by the CA line to permit SRQ to be asserted over the HP-IB bus. If CA is high, the HP-85 monitors SRQ from the bus. If CA is low, the HP-85 pulls SRQ when it requests service from another controller.
REN and IFC	Two send/receive pairs are enabled to send or receive the state of these bus lines in the same direction simultaneously.

---

* CONTROLLER ACTIVE (CA) is synonymous with the term controller-in-charge which is referenced in IEEE Standard 488-1978.

8049 Microcomputer (μC)

Most of the activities carried out by the 8049 μC have already been discussed in this section. This discussion will summarize these activities, give the pin assignment and provide register tables.

The μC is the intermediary between the HP-85 and the HP-IB bus. It implements interface protocol via its own self contained ROM. The μC responds to instructions from the host HP-85 or the bus, depending upon which bus device is assigned controller status. When the host HP-85 is system controller, the μC can recognize an interrupt from an HP-IB bus device, and, depending upon user programmed interrupt conditions, it may service the interrupt or it may in turn interrupt the HP-85.

There are two 8-bit I/O ports that connect the μC to the HP-IB bus. Port 1 connects to the handshake and control lines; port 2 connects to the data lines (DIO1 through DIO8). There are eight other bits (D0 through D7) which go to the address-data bus connecting the μC to the Translator.

The following table lists all the lines used on the 8049 μC.

Table 3-2. 8049 Microcomputer Pin Assignment

Designation	Pin Number	Function
V_{SS}	20	Circuit GND.
V_{DD}	26	+5V.
V_{CC}	40	+5V.
P10 – P17 (Port 1)	27 – 34	8-bit I/O port that connects to the control and handshake lines on the HP-IB bus.
P20 – P27 (Port 2)	21 – 24 35 – 38	8-bit I/O port that connects to the data lines (DIO1 – DIO8) on the HP-IB bus.
D0 – D7 (Address-Data Bus)	12 – 19	A bi-directional port that connects to the address-data bus from the Translator.
T0	1	An input used to sample ATN. If ATN is true, the μC remains in interrupt service routine and handshakes commands. If ATN is false, the μC leaves the interrupt service routine and returns to the main program.
T1	39	An input which initiates an interrupt of the μC.
$\overline{\text{INT}}$	6	Interrupt input. Initiates an interrupt to the HP-85 from the HP-IB bus if interrupt is enabled. Interrupt is disabled after a reset (active low.
$\overline{\text{RD}}$	8	Read strobe. When low the μC reads the address-data bus (D0 – D7).
$\overline{\text{WR}}$	10	Write strobe. When low the μC writes to the address-data bus (D0 – D7).
$\overline{\text{RESET}}$	4	Reset. An active low strobe which is used to initialize the μC.
ALE	11	Address Latch Enable. The negative edge of ALE causes an address the μC has placed on the address-data bus (D0 – D7) to be latched into the Translator.
XTAL 1	2	One side of crystal input for internal oscillator.
XTAL 2	3	Other side of crystal input.
$\overline{\text{PSEN}}$	9	Not used.
EA	7	Circuit GND.
$\overline{\text{SS}}$	5	Not used.

Internal to the μC are seven 8-bit status registers and twelve 8-bit control registers. The status registers indicate the state of the μC at a given time. The control registers are written to when the state of the interface is to be changed. Register tables of the status and control registers are given in Figures 3-7 and 3-8. If a detailed description of these registers is desired, refer to the I/O ROM manual.

Register Number	**Bit Number**								**Register Function**
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
SRQ	0	0	0	0	0	0	0	1	Interface Identification
SR1	IFC	LA	CA	TA	SRQ	DCL or SDC	GET	SCG	Interrupt Cause
SR2	IFC	REN	SRQ	ATN	EOI	DAV	NDAC	NRFD	HP-IB Control Lines
SR3	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1	HP-IB Data Lines
SR4	0	0	System Controller	A4	A3	A2	A1	A0	HP-IB Address and System Controller
SR5	System Controller	LA	CA	TA	SPE	Parity Error	Remote	Local Lockout	State Register
SR6	0	0	0	SC5	SC4	SC3	SC2	SC1	Secondary Commands

Figure 3-7. μC Status Registers

CAUTION

Do not write to Control Registers 0 through 3 unless you have an I/O ROM and you are completely familiar with the function of these registers. In particular, Control Registers 2 and 3 provide direct access to the HP-IB control and data lines. They must be used with care, and used only by persons aware of HP-IB protocols! It is possible to cause a bus malfunction or device damage by improper use of these registers.

Register Number	Bit Number								Register Function
	7	6	5	4	3	2	1	0	
CR0	X	X	X	X	Odd Parity	Even Parity	One Parity	Zero Parity	Parity Control
CR1	IFC Mask	LA Mask	CA Mask	TA Mask	SRQ Mask	Clear Mask	Trigger Mask	SCG Mask	Interrupt Mask
CR2	X	REN	SRQ	ATN	EOI	DAV	DAC	RFD	HP-IB Control Lines
CR3	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1	HP-IB Data Lines
CR16	EOI Enable	X	X	X	X	EOL2	EOL1	EOL0	EOL Control
CR17				Default = CR					Character #1
CR18				Default = LF					Character #2
CR19									Character #3
CR20									Character #4
CR21									Character #5
CR22									Character #6
CR23									Character #7

Figure 3-8. μC Control Registers

Table 3-3. Replaceable Parts List

Reference Designator	HP Part No.	TQ	Description
A1	82937-60901	1	PC Assembly
C1, C2	0180-0228	2	C-F: 22μF, 15V
C3-C13	0160-0576	12	C-F: 0.1μF, 50V
C14	0160-4767	1	C-F: 20pF, 200V
C15	0160-0576	0	C-F: 0.1μF, 50V
J1	1251-5266	1	Connector: 24 Pin
MP1	0340-0883	1	Transistor Insulator
Q1	1854-0019	1	Transistor: NPN, Si
R1	0683-1035	1	R-F: 10kΩ, 5%, .25W
R2, R3	0683-2225	2	R-F: 2.2kΩ, 5%, .25W
R4	0683-1025	1	R-F: 1kΩ, 5%, .25W
R5, R6	1810-0278	2	Resistor Network: 3.3kΩ, 2%, 1.25W
R7	1810-0367	1	Resistor Network: 4.7kΩ, 2%, 1.25W
S1	3101-0491	1	Switch: 10 Segment SPDT
U1	1MB5-0101	1	IC: Translator
U2	1820-2437	1	IC: 8049 Microcomputer
U3	1820-1466	1	IC: 74C174, Hex D Flip-Flop
U4	1820-1402	1	IC: 80C95, Hex Buffer
U5	1820-1198	1	IC: 74LS03, Quad 2-Input NAND
U6	1820-1416	1	IC: 74LS14, Hex Schmitt Trigger Inverter
U7	1820-1197	1	IC: 74LS00, Quad 2-Input NAND
U8	1820-1144	1	IC: 74LS02, Quad 2-Input NOR
U9, U10	1820-2424	2	IC: MC3447, Bi-directional Bus Transceivers
U11	1820-1112	1	IC: 74LS74, Quad D Flip-Flop
W1	8150-3773	1	Wire Jumper
Y1	0410-1222	1	11 MHz Crystal
	82937-60902	1	Case
	82937-60004	1	Interface Cable
	0590-0199	2	Hex Nut with Lock Washer
	1400-1063	1	Top Cable Clamp
	1400-1064	1	Bottom Cable Clamp
	2200-0143	6	Screws: 4-40 Machine
	0363-0174	1	Ground Contact

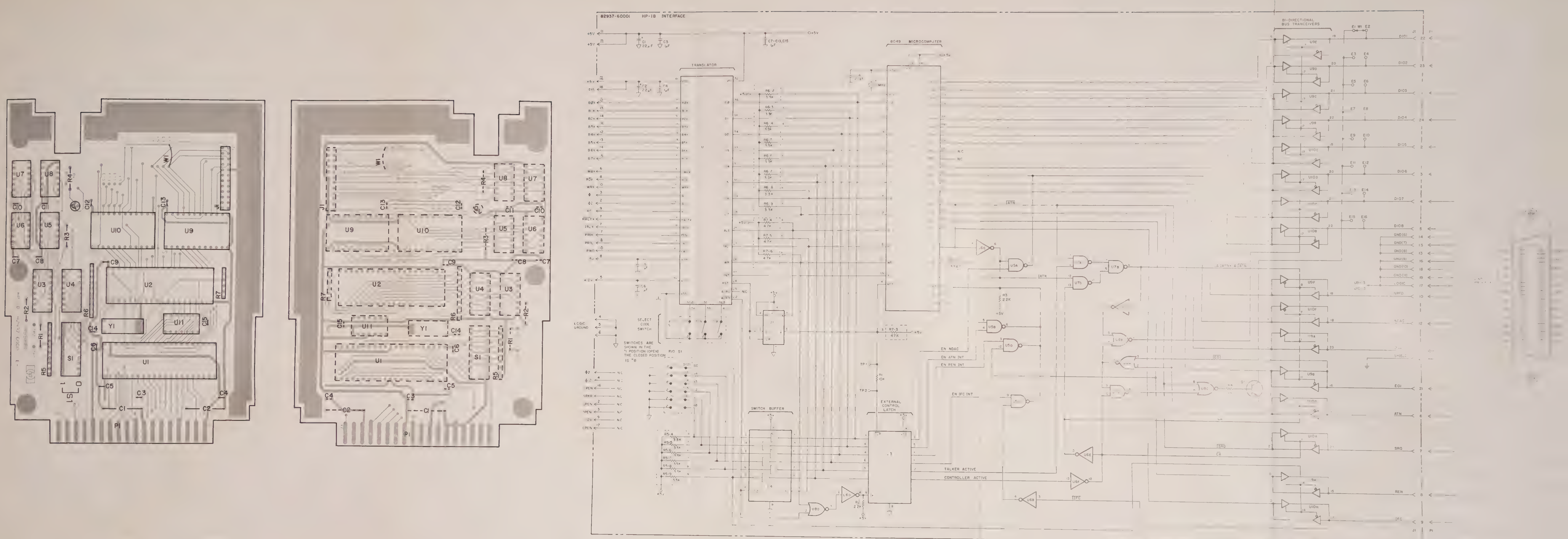

U7
U8
R4
W1
C10
C11
C12
C13
J1
U6
U5
R3
U10
U9
C7
C8
C9
U3
U4
U2
R2
R7
R6
C14
Y1
U11
C15
R1
S1
C6
U1
R5
C5
C3
C4
C1
C2
P1
R9
82937-60001 HP-IB INTERFACE
TRANSLATOR
8049 MICROCOMPUTER
BI-DIRECTIONAL BUS TRANCEIVERS
SELECT CODE SWITCH
SWITCHES ARE SHOWN IN THE "1" POSITION (OPEN) THE CLOSED POSITION IS "0"
LOGIC GROUND
SWITCH BUFFER
EXTERNAL CONTROL LATCH
EN NDAC
EN ATN INT
EN REN INT
EN IFC INT
TALKER ACTIVE
CONTROLLER ACTIVE
ATN
DIO1
DIO2
DIO3
DIO4
DIO5
DIO6
DIO7
DIO8
NDAC
EOI
ATN
SRQ
REN
IFC
TP1
TP2
+5V

Figure 3-9. Schematic Diagram

SALES OFFICES

Arranged alphabetically by country

ANGOLA
Telectra
Empresa Técnica de Equipamentos Eléctricos, S A.R L
R Barbosa Rodrigues, 41-I°DT °
Caixa Postal, 6487
Luanda
Tel 35515/6

ARGENTINA
Hewlett-Packard Argentina S.A
Santa Fe 2035, Martinez
6140 **Buenos Aires**
Tel. 792-1239, 798-6086
Telex 122443 AR CIGY

Biotron S.A.C.I y M
Avda Paseo Colon 221
9 piso
1399 **Buenos Aires**
Tel 30-4846/1851/8384
34-9356/0460/4551
Telex. (33) 17595 BIO AR

AUSTRALIA

AUSTRALIA CAPITAL TERR.
Hewlett-Packard Australia Pty Ltd
121 Wollongong Street
Fyshwick, 2609
Tel 804244
Telex 62650

NEW SOUTH WALES
Hewlett-Packard Australia Pty Ltd
31 Bridge Street
Pymble, 2073
Tel 4496566
Telex 21561

QUEENSLAND
Hewlett Packard Australia Pty Ltd
5th Floor
Teachers Union Building
495-499 Boundary Street
Spring Hill, 4000
Tel 2291544

SOUTH AUSTRALIA
Hewlett-Packard Australia Pty Ltd
153 Greenhill Road
Parkside, 5063
Tel 2725911
Telex 82536

VICTORIA
Hewlett-Packard Australia Pty Ltd
31-41 Joseph Street
Blackburn, 3130
Tel 89-6351
Telex 31024 MELB

WESTERN AUSTRALIA
Hewlett-Packard Australia Pty Ltd
141 Stirling Highway
Nedlands, 6009
Tel 3865455
Telex 93859

AUSTRIA
Hewlett-Packard Ges m b H
Wehlistrasse 29
P O Box 7
A 1205 **Vienna**
Tel 35-16-21-0
Telex 13582/135066

Hewlett Packard Ges m b H
Wehlistrasse, 29
A 1205 **Wien**
Tel 35-16-21
Telex 135066

BAHRAIN
Medical Only
Wael Pharmacy
P O Box 648
Bahrain
Tel 54886 56123
Telex 8550 WAEL GJ

Al Hamidiya Trading and Contracting
P O Box 20074
Manama
Tel 259978 259958
Telex 8895 KALDIA GJ

BANGLADESH
The General Electric Co of Bangladesh Ltd
Magnet House 72
Dilkusha Commercial Area
Motijheel, Dacca 2
Tel 252415 252419
Telex 734

BELGIUM
Hewlett Packard Benelux S A /N V
Avenue du Col Vert, 1
(Groenkraaglaan)
B 1170 **Brussels**
Tel (02) 660 50 50
Telex 23 494 paloben bru

BRAZIL
Hewlett-Packard do Brasil I.e C Ltda
Alameda Rio Negro, 750
Alphaville
06400 **Barueri** SP
Tel 429 3222

Hewlett-Packard do Brasil I.e C Ltda.
Rua Padre Chagas, 32
90000-**Pôrto Alegre**-RS
Tel: 22-2998, 22-5621

Hewlett-Packard do Brasil I e C Ltda
Av Epitacio Pessoa, 4664
22471-**Rio de Janeiro**-RJ
Tel 286-0237
Telex 021-21905 HPBR-BR

CANADA

ALBERTA
Hewlett-Packard (Canada) Ltd.
11620A 168th Street
Edmonton T5M 3T9
Tel (403) 452-3670
TWX 610-831-2431

Hewlett-Packard (Canada) Ltd
210, 7220 Fisher St S E
Calgary T2H 2H8
Tel (403) 253-2713
TWX 610-821-6141

BRITISH COLUMBIA
Hewlett-Packard (Canada) Ltd
10691 Shellbridge Way
Richmond V6X 2W7
Tel (604) 270-2277
TWX 610-925-5059

MANITOBA
Hewlett-Packard (Canada) Ltd
380-550 Century St
St. James,
Winnipeg R3H 0Y1
Tel (204) 786-6701
TWX 610-671-3531

NOVA SCOTIA
Hewlett-Packard (Canada) Ltd
P O Box 931
800 Windmill Road
Dartmouth B3B 1L1
Tel (902) 469-7820
TWX 610-271-4482

ONTARIO
Hewlett-Packard (Canada) Ltd
1020 Morrison Dr
Ottawa K2H 8K7
Tel (613) 820-6483
TWX 610-563-1636

Hewlett-Packard (Canada) Ltd
6877 Goreway Drive
Mississauga L4V 1M8
Tel. (416) 678-9430
TWX 610-492 4246

Hewlett-Packard (Canada) Ltd
552 Newbold Street
London N6E 2S5
Tel (519) 686-9181
TWX 610-352-1201

QUEBEC
Hewlett-Packard (Canada) Ltd
275 Hymus Blvd
Pointe Claire H9R 1G7
Tel (514) 697-4232
TWX 610-422 3022

FOR CANADIAN AREAS NOT LISTED:
Contact Hewlett Packard (Canada) Ltd in Mississauga

CHILE
Jorge Calcagni y Cia Ltda
Arturo Burhle 065
Casilla 16475
Correo 9, **Santiago**
Tel 220222
Telex JCALCAGNI

COLOMBIA
Instrumentación
Henrik A Langebaek & Kier S A
Carrera 7 No 48-75
Apartado Aéreo 6287
Bogotá, 1 D E
Tel 269-8877
Telex 44400

Instrumentación
H A Langebaek & Kier S A
Carrera 63 No 49-A 31
Apartado 54098
Medellin
Tel 304475

COSTA RICA
Cientifica Costarricense S A
Avenida 2 Calle 5
San Pedro de Montes de Oca
Apartado 10159
San José
Tel 24 38 20 24 08 19
Telex 2367 GALGUR CR

CYPRUS
Kypronics
19 Gregorios Xenopoulos Street
P O Box 1152
Nicosia
Tel 45628/29
Telex 3018

CZECHOSLOVAKIA
Hewlett-Packard
Obchodni zastupitelstvi v CSSR
Pisemny styk
Post schranka 27
CS 118 01 **Praha** 011
CSSR

Vyvojova a Provozni Zakladna Vyzkumnych Ustavu v Bechovicich
CSSR-25097 **Bechovice u Prahy**
Tel 89 93 41
Telex 12133

Institute of Medical Bionics
Vyskumny Ustav Lekarskej Bioniky
Jedlova 6
CS 88346 **Bratislava-Kramare**
Tel 44-551
Telex 93229

DENMARK
Hewlett-Packard A/S
Datavej 52
DK 3460 **Birkerod**
Tel (02) 81 66 40
Telex 37409 hpas dk

Hewlett-Packard A/S
Navervej 1
DK 8600 **Silkeborg**
Tel (06) 82 71 66
Telex 37409 hpas dk

ECUADOR
CYEDE Cia Ltda
P O Box 6423 CCI
Av. Eloy Alfaro 1749
Quito
Tel 450-975, 243-052
Telex 2548 CYEDE ED

Medical Only
Hospitalar S A
Casilla 3590
Robles 625
Quito
Tel 545-250

EGYPT
I E A
International Engineering Associates
24 Hussein Hegazi Street
Kasr-el-Aini
Cairo
Tel 23 829
Telex 93830

SAMITRO
Sami Amin Trading Office
18 Abdel Aziz Gawish
Abdine-Cairo
Tel 24932

EL SALVADOR
IPESA
Bulevar de los Heroes 11-48
Edificio Sarah 1148
San Salvador
Tel 252787

ETHIOPIA
Abdella Abdulmalik
P O Box 2635
Addis Ababa
Tel 11 93 40

FINLAND
Hewlett-Packard Oy
Revontulentie, 7
SF-02100 **Espoo** 10
Tel (90) 455 0211
Telex 121563 hewpa sf

FRANCE
Hewlett-Packard France
Zone d'activites de Courtaboeuf
Avenue des Tropiques
Boite Postale 6
91401 **Orsay** Cédex
Tel (1) 907 78 25
TWX 600048F

Hewlett Packard France
Chemin des Mouilles
B P 162
69130 **Ecully**
Tel (78) 33 81 25
TWX 310617F

Hewlett Packard France
20, Chemin de La Cépière
31081 **Toulouse Le Mirail**-Cédex
Tel (61) 40 11 12

Hewlett Packard France
Le Ligoures
Place Romée de Villeneuve
13100 **Aix-en-Provence**
Tel (42) 59 41 02
TWX 410770F

Hewlett Packard France
2, Allee de la Bourgonette
35100 **Rennes**
Tel (99) 51 42 44
TWX 740912F

Hewlett-Packard France
18, rue du Canal de la Marne
67300 **Schiltigheim**
Tel (88) 83 08 10
TWX 890141F

Hewlett-Packard France
Immeuble péricentre
rue van Gogh
59650 **Villeneuve** D'Ascq
Tel (20) 91 41 25
TWX 160124F

Hewlett-Packard France
Bâtiment Ampère
Rue de la Commune de Paris
B P 300
93153 **Le Blanc Mesnil**-Cédex
Tel. (01) 931 88 50
Telex 211032F

Hewlett-Packard France
Av du Pdt. Kennedy
33700 **Merignac**
Tel (56) 97 01 81

Hewlett-Packard France
Immeuble Lorraine
Boulevard de France
91035 **Evry**-Cédex
Tel. 077 96 60
Telex 692315F

Hewlett-Packard France
23 Rue Lothaire
57000 **Metz**
Tel (87) 65 53 50

GERMAN FEDERAL REPUBLIC
Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Berner Strasse 117
Postfach 560 140
D 6000 **Frankfurt** 56
Tel (06011) 50041
Telex 04 13249 hpffm d

Hewlett-Packard GmbH
Technisches Büro Böblingen
Herrenberger Strasse 110
D-7030 **Böblingen**, Wurttemberg
Tel (07031) 667-1
Telex 07265739 bbn

Hewlett-Packard GmbH
Technisches Büro Düsseldorf
Emanuel-Leutze Str 1 (Seestern)
D 4000 **Düsseldorf**
Tel (0211) 5971-1
Telex 085/86 533 hpdd d

Hewlett-Packard GmbH
Technisches Büro Hamburg
Kapstadtring 5
D-2000 **Hamburg** 60
Tel (040) 63804 1
Telex 21 63 032 hphh d

Hewlett-Packard GmbH
Technisches Büro Hannover
Am Grossmarkt 6
D 3000 **Hannover** 91
Tel (0511) 46 60 01
Telex 092 3259

Hewlett-Packard GmbH
Technisches Büro Nürnberg
Neumeyerstrasse 90
D 8500 **Nürnberg**
Tel (0911) 52 20 83
Telex 0623 860

Hewlett Packard GmbH
Technisches Büro München
Eschenstrasse 5
D-8021 **Taufkirchen**
Tel (089) 6117-1
Telex 0524985

Hewlett-Packard GmbH
Technisches Büro Berlin
Kaithstrasse 2-4
D 1000 **Berlin** 30
Tel (030) 24 90 86
Telex 018 3405 hpbln d

GREECE
Kostas Karayannis
8 Omirou Street
Athens 133
Tel 32 30 303.32/37 731
Telex 21 59 62 RKAR GR

GUAM
Guam Medical Supply, Inc
Suite C, Airport Plaza
P O Box 8947
Tamuning 96911
Tel 646-4513

GUATEMALA
IPESA
Avenida Reforma 3-48
Zona 9
Guatemala City
Tel 316627, 314786, 66471-5, ext 9
Telex 4192 Teletro Gu

HONG KONG
Hewlett-Packard Hong Kong Ltd
11th Floor, Four Seas Bldg
212 Nathan Rd
Kowloon
Tel 3 697446 (5 lines)
Telex 36678 HX

Medical/Analytical Only
Schmidt & Co (Hong Kong) Ltd
Wing On Centre, 28th Floor
Connaught Road, C
Hong Kong
Tel 5-455644
Telex 74766 SCHMX HX

INDIA
Blue Star Ltd
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
Bombay 400 025
Tel 45 78 87
Telex 011-4093

Blue Star Ltd
Band Box House
Prabhadevi
Bombay 400 025
Tel 45 73 01
Telex 011-3751

Blue Star Ltd
Bhavdeep
Stadium Road
Ahmedabad 380 014
Tel 43922
Telex 012-234

Blue Star Ltd
7 Hare Street
Calcutta 700 001
Tel 23-0131
Telex 021-7655

Blue Star Ltd
Bhandari House
91 Nehru Place
New Delhi 110 024
Tel 682547
Telex 031 2463

Blue Star Ltd
T C 7/603 'Poornima'
Maruthankuzhi
Trivandrum 695 013
Tel 65799
Telex 0884 259

Blue Star Ltd
11 Magarath Road
Bangalore 560 025
Tel 55668
Telex 0845-430

Blue Star Ltd
Meeakshi Mandiram
XXXXV/1379-2 Mahatma Gandhi Rd
Cochin 682 016
Tel 32069
Telex 085-514

Blue Star Ltd
1-1-117/1 Sarojini Devi Road
Secunderabad 500 033
Tel 70126
Telex 0155 459

Blue Star Ltd
133 Kodambakkam High Road
Madras 600 034
Tel 82057
Telex 041-379

ICELAND
Medical Only
Elding Trading Company Inc
Hafnarnvoli - Tryggvagötu
P O Box 895
IS-**Reykjavik**
Tel 1 58 20/1 63 03

INDONESIA
BERCA Indonesia P T
P O Box 496/Jkt
Jln Abdul Muis 62
Jakarta
Tel 349255 349886
Telex 46748 BERSIL IA

BERCA Indonesia P T
P O Box 174/Sby
23 Jln Jimerto
Surabaya
Tel 42027

IRELAND
Hewlett-Packard Ltd
Kestrel House
Clanwilliam Place
Lower Mount Street
Dublin 2, Eire

Hewlett-Packard Ltd
2C Avongberg Ind Est
Long Mile Road
Dublin 12
Tel 514322/514224
Telex 30439

Medical Only
Cardiac Services (Ireland) Ltd
Kilmore Road
Artane
Dublin 5, Eire
Tel (01) 315820

Medical Only
Cardiac Services Co
95A Finaghy Rd South
Belfast BT10 0BY
GB Northern Ireland
Tel (0232) 625566
Telex 747626

ISRAEL
Electronics Engineering Div of Motorola Israel Ltd
16, Kremenetski Street
P O Box 25016
Tel-Aviv
Tel 38973
Telex 33569 34164

ITALY
Hewlett Packard Italiana S p A
Via G Di Vittorio, 9
20063 **Cernusco Sul Naviglio** (MI)
Tel (2) 903691
Telex 334632 HEWPACKIT

Hewlett-Packard Italiana S p A
Via Turazza, 14
35100 **Padova**
Tel (49) 664888
Telex 430315 HEWPACKI

Hewlett-Packard Italiana S p A
Via G Armellini 10
I 00143 **Roma**
Tel (06) 54 69 61
Telex 610514

Hewlett-Packard Italiana S p A
Corso Giovanni Lanza 94
I-10133 **Torino**
Tel (011) 659308
Telex 221079

Hewlett-Packard Italiana S p A
Via Principe Nicola 43 G/C
I-95126 **Catania**
Tel (095) 37 05 04
Telex 970291

Hewlett-Packard Italiana S p A
Via Nuova san Rocco A Capadimonte, 62A
80131 **Napoli**
Tel (081) 710698

Hewlett-Packard Italiana S p A
Via Martin Luther King, 38/111
I-40132 **Bologna**
Tel (051) 402394
Telex 511630

JAPAN
Yokogawa Hewlett-Packard Ltd
29-21, Takaido Higashi 3 chome
Suginami ku, **Tokyo** 168
Tel 03 331-6111
Telex 232 2024 YHP Tokyo

Yokogawa Hewlett-Packard Ltd
Chuo Bldg 4th Floor
4 20, Nishinakajima 5 chome
Yodogawa-ku, Osaka shi
Osaka, 532
Tel 06 304 6021
Telex 523 3624

Yokogawa-Hewlett Packard Ltd
Sumitomo Seimei Nagaya Bldg
11 2 Shimosasajima cho,
Nakamura-ku **Nagoya**, 450
Tel 052 571 5171

Yokogawa Hewlett Packard Ltd
Tanigawa Building
2 24 1 Tsuruya cho
Kanagawa ku
Yokohama, 221
Tel 045 312 1252
Telex 382 3204 YHP YOK

Yokogawa-Hewlett-Packard Ltd
Mito Mitsui Building
105, 1 chome, San-no-maru
Mito, Ibaragi 310
Tel 0292-25-7470

Yokogawa-Hewlett-Packard Ltd
Inoue Building
1348-3, Asahi-cho, 1-chome
Atsugi, Kanagawa 243
Tel 0462 24 0452

Yokogawa Hewlett-Packard Ltd
Kumagaya Asahi
Hachijuni Building
4th Floor
3 4, Tsukuba
Kumagaya, Saitama 360
Tel 0485-24-6563

JORDAN
Mouasher Cousins Co
P O Box 1387
Amman
Tel 24907/39907
Telex SABCO JO 1456

KENYA
ADCOM Ltd , Inc
P O Box 30070
Nairobi
Tel 331955
Telex 22639

Medical Only
International Aeradio (E A) Ltd
P O Box 19012
Nairobi Airport
Nairobi
Tel 336055/56
Telex 22201/22301

Medical Only
International Aeradio (E A) Ltd
P O Box 95221
Mombasa

KOREA
Samsung Electronics Co , Ltd
4759 Shingil-6-Dong
Yeong Deung POU
Seoul
Tel 833-4122, 4121
Telex SAMSAN 27364

KUWAIT
Al-Khaldiya Trading & Contracting
P O Box 830 Salat
Kuwait
Tel 42 4910/41 1726
Telex 2481 Areeg kt

LUXEMBURG
Hewlett-Packard Beneluz S A /N V
Avenue du Col-Vert, 1
(Groenkraaglaan)
B-1170 **Brussels**
Tel (02) 660 5050
Telex 23 494

MALAYSIA
Hewlett Packard Sales (Malaysia) Sdn Bhd
Suite 2 21/2 22
Bangunan Angkasa Raya
Jalan Ampang
Kuala Lumpur
Tel 483680 485653

Protel Engineering
P O Box 1917
Lot 259 Satok Road
Kuching **Sarawak**
Tel 53544

MEXICO
Hewlett Packard Mexicana S A de C V
Av Periférico Sur No 6501
Tepepan, Xochimilco
Mexico 23, D F
Tel 905 676 4600
Telex 017-74-507

Hewlett Packard Mexicana S A de C V
Rio Volga #600
Col Del Valle
Monterrey, N L
Tel 78-32 10

MOROCCO
Dolbeau
81 rue Karatchi
Casablanca
Tel 3041 82
Telex 23051 22822

Gerep
2 rue d'Agadir
Boite Postal 156
Casablanca
Tel 272093.5
Telex 23 739

MOZAMBIQUE
A N Goncalves, Ltd
162, 1° Apt 14 Av D Luis
Caixa Postal 107
Maputo
Tel 27091, 27114
Telex 6-203 NEGON Mo

NETHERLANDS
Hewlett-Packard Benelux N V
Van Heuven Goedhartlaan 121
P O Box 667
1181KK **Amstelveen**
Tel (20) 47 20 21
Telex 13 216

NEW ZEALAND
Hewlett-Packard (N Z) Ltd
4-12 Cruickshank Street
Kilbirnie, Wellington 3
P O Box 9443
Courtney Place
Wellington
Tel 877 199

Hewlett-Packard (N Z) Ltd
P O Box 26 189
169 Manukau Road
Epsom, **Auckland**
Tel 687 159

Analytical/Medical Only
Northrop Instruments & Systems Ltd ,
Sturdee House
85-87 Ghuznee Street
P O Box 2406
Wellington
Tel 850-091
Telex NZ 31291

Northrup Instruments & Systems Ltd
Eden House, 44 Khyber Pass Rd
P O Box 9682 Newmarket
Auckland 1
Tel 794-091

Northrup Instruments & Systems Ltd
Terrace House 4 Oxford Terrace
P O Box 8388
Christchurch
Tel 64-165

NIGERIA
The Electronics Instrumentations Ltd
N6B/770 Oyo Road
Oluseun House
P M B 5402
Ibadan
Tel 461577
Telex 31231 TEIL NG

The Electronics Instrumentations Ltd
144 Agege Motor Road, Mushin
P O Box 481
Mushin **Lagos**

NORWAY
Hewlett-Packard Norge A/S
Ostendalen 18
P O Box 34
1345 **Osteraas**
Tel (02) 1711 80
Telex 16621 hpnas n

Hewlett-Packard Norge A/S
Nygaardsgaten 114
P O Box 4210
5013 Nygaardsgaten
Bergen
Tel (05) 21 97 33

PANAMA
Electrónico Balboa S A
Aparatado 4929
Panama 5
Calle Samuel Lewis
Edificio "Alfa," No 2
Ciudad de Panama
Tel 64 2700
Telex 3483103 Curundu, Canal Zone

PERU
Compañía Electro Médica S A
Los Flamencos 145
San Isidro Casilla 1030
Lima 1
Tel 41 4325
Telex Pub Booth 25424 SISIDRO

PAKISTAN
Mushko & Company Ltd
Oosman Chambers
Abdullah Haroon Road
Karachi 3
Tel 511027 512927
Telex 2894

SALES OFFICES

Arranged alphabetically by country (cont.)

Mushko & Company Ltd
[illegible] Bazar Rd
Sector G 6 4
Islamabad
Tel 28264

PHILIPPINES

The Online Advanced Systems Corporation
Rico House
Amorsolo cor Herrera Str
Legaspi Village Makati
P O Box 1510
Metro **Manila**
Tel 85 35 81 85-34 91
85 32 21
Telex 3274 ONLINE

RHODESIA

Field Technical Sales
45 Kelvin Road North
P O Box 3458
Salisbury
Tel 705231 (5 lines)
Telex RH 4122

POLAND

Biuro Informacji Technicznej
Hewlett Packard
Ul Slawki 2, 6P
PL00 950 **Warszawa**
Tel 39 59 62, 39 51 87
Telex 81 24 53

PORTUGAL

Telectra-Empresa Técnica de
Equipamentos Eléctricos
S a r l
Rua Rodrigo da Fonseca 103
P O Box 2531
P **Lisbon** 1
Tel (19) 68 60 72
Telex 12598

Medical Only
Mundinter
Intercambio Mundial de
Comércio S a r l
P O Box 2761
Avenida Antonio Augusto
de Aguiar 138
P **Lisbon**
Tel (19) 53 21 31/7
Telex 16691 munter p

PUERTO RICO

Hewlett Packard Inter
Americas
Puerto Rico Branch Office
Calle 272
#203 Urb Country Club
Carolina 00630
Tel (809) 762 7255
Telex 345 0514

QATAR

Nasser Trading & Contracting
P O Box 1563
Doha
Tel 22170
Telex 4439 NASSER

ROMANIA

Hewlett Packard
Reprezentanta
Bd n Balcescu 16
Bucuresti
Tel 15 80 23/13 88 85
Telex 10440

SAUDI ARABIA

Modern Electronic
Establishment Head Office
P O Box 1228 Baghdadiah
Street
Jeddah
Tel [illegible] 798
Telex [illegible]
Cable ELECTA JEDDAH

Modern Electronic
Establishment Branch
P O Box [illegible]
Riyadh
Tel [illegible]
Telex [illegible]

Modern Electronic
Establishment Branch
P O Box [illegible]
Al-Khobar
Tel [illegible]
Telex [illegible]
Cable ELECTA AL-KHOBAR

SINGAPORE

Hewlett Packard Singapore
Pte Ltd
6th Floor Inchcape House
450-452 Alexandra Road
P O Box 58
Alexandra Post Office
Singapore [illegible]
Tel [illegible]
Telex HPSG RS [illegible]486

SOUTH AFRICA

Hewlett Packard South Africa
(Pty), Ltd
Private Bag Wendywood,
Sandton, Transvaal, 2144
Hewlett Packard Centre
Daphne Street, Wendywood,
Sandton, 2144
Tel 802-5111/25
Telex 8 4782

Hewlett Packard South Africa
(Pty), Ltd
P O Box 120
Howard Place
Cape Province 7450
Pine Park Centre Forest Drive
Pinelands,
Cape Province, 7405
Tel 53 7955 thru 9
Telex 57 0006

SPAIN

Hewlett-Packard Española
S A
Calle Jerez 3
E **Madrid** 16
Tel (1) 458 26 00 (10 lines)
Telex 23515 hpe

Hewlett-Packard Española S A
Colonia Mirasierra
Edificio Juban
c/o Costa Brava 13
Madrid 34

Hewlett-Packard Española
S A
Milanesado 21 23
E **Barcelona** 17
Tel (3) 203 6200 (5 lines)
Telex 52603 hpbe e

Hewlett Packard Española
S A
Av Ramón y Cajal, 1
Edificio Sevilla planta 9°
E **Sevilla** 5
Tel 64 44 54/58

Hewlett-Packard Española S A
Edificio Albia II 7° B
E **Bilbao** 1
Tel 23 83 06/23 82 06

Hewlett Packard Española S A
C/Ramon Gordillo 1
(Entlo)
E **Valencia** 10
Tel 96 361 13 54/361 13 58

SRI LANKA

Metropolitan Agencies Ltd
209/9 Union Place
Colombo 2
Tel 35947
Telex 1377METROLTD CE

SUDAN

Radison Trade
P O Box 921
Khartoum
Tel 44048
Telex 375

SURINAM

Surtel Radio Holland N V
Grote Holstr 3 5
P O Box 155
Paramaribo
Tel 72118 77880

SWEDEN

Hewlett Packard Sverige AB
Enighetsvägen 3 Fack
S 161 **Bromma** 20
Tel (08) 730 05 50
Telex 10721
Cable MEASUREMENTS
Stockholm

Hewlett-Packard Sverige AB
Frötallisgatan 30
S 4[illegible] **Västra**
Frölunda
Tel (031) 49 09 50
Telex 10721 via Bromma
office

SWITZERLAND

Hewlett Packard (Schweiz) AG
Zürcherstrasse 20
P O Box 307
CH 8952 **Schlieren-**
Zurich
Tel (01) 7305240
Telex 53933 hpag ch
Cable HPAG CH

Hewlett Packard (Schweiz) AG
Château Bloc 19
CH 1219 **Le Lignon-**
Geneva
Tel (022) 96 03 22
Telex 27333 hpag ch
Cable HEWPACKAG Geneva

SYRIA

General Electronic Inc
Nuri Basha Ahnaf Ebn Kays
Street
P O Box 5781
Damascus
Tel 33 24 87
Telex 11215 ITIKAL
Cable ELECTROBOR
DAMASCUS

Medical only
Sawah & Co
Place Azmé
B P 2308
Damascus
Tel 16 367-19 697 14 268
Telex 11304 SATACO SY
Cable SAWAH, DAMASCUS

Suleiman Hilal El Mlawi
P O Box 2528
Mamoun Bitar Street, 56 58
Damascus
Tel 11 46 63
Telex 11270
Cable HILAL DAMASCUS

TAIWAN

Hewlett Packard Far East Ltd
Taiwan Branch
Bank Tower, 5th Floor
205 Tun Hau North Road
Taipei
Tel (02) 751-0404 (15 lines)

Hewlett Packard Far East Ltd
Taiwan Branch
68-2 Chung Cheng 3rd Road
Kaohsiung
Tel (07) 242318-Kaohsiung

Analytical Only
San Kwang Instruments Co ,
Ltd
20 Yung Sui Road
Taipei
Tel 3615446 9 (4 lines)
Telex 22894 SANKWANG

TANZANIA

Medical Only
International Aeradio (E A), Ltd
P O Box 861
Dar es Salaam
Tel 21251 Ext 265
Telex 41030

THAILAND

UNIMESA Co Ltd
Elcom Research Building
2538 Sukumvit Ave
Bangchak, Bangkok
Tel 39 32 387 39 30 338

TRINIDAD & TOBAGO

CARTEL
Caribbean Telecoms Ltd
P O Box 732
69 Frederick Street
Port-of-Spain
Tel 62 53068

TUNISIA

Tunisie Electronique
31 Avenue de la Liberte
Tunis
Tel 280 144

Corema
1 ter Av de Carthage
Tunis
Tel 253 821
Telex 12319 CABAM TN

TURKEY

TEKNIM Company Ltd
Riza Sah Pehlevi
Caddesi No 7
Kavaklidere **Ankara**
Tel 275800
Telex 42155

Teknim Com Ltd
Barbaros Bulvari 55, 12
Besiktas **Istanbul**
Tel 613 546
Telex 23540

E M A
Muhendislik Kollektif Sirketi
Mediha Eldem Sokak 41/6
Yuksel Caddesi
Ankara
Tel 17 56 22

Yilmaz Ozyurek
Milli Mudafaa Cad 16/6
Kizilay
Ankara
Tel 25 03 09 17 80 26
Telex 42576 OZEK TR

UNITED ARAB EMIRATES

Emitac Ltd (Head Office)
P O Box 1641
Sharjah
Tel 354121 3
Telex 8136

Emitac Ltd (Branch Office)
P O Box 2711
Abu Dhabi
Tel 331370, 1

UNITED KINGDOM

Hewlett Packard Ltd
King Street Lane
Winnersh, Wokingham
Berkshire RG11 5AR
GB England
Tel (0734) 784774
Telex 84 71 78/9

Hewlett Packard Ltd
Fourier House
257 263 High Street
London Colney
St. Albans, Herts
GB-England
Tel (0727) 24400
Telex 1 8952716

Hewlett Packard Ltd
Trafalgar House
Navigation Road
Altrincham
Cheshire WA14 1NU
GB England
Tel (061) 928 6422
Telex 668068

Hewlett Packard Ltd
Lygon Court
Hereward Rise
Dudley Road
Halesowen,
West Midlands B62 8SD
GB-England
Tel (021) 501 1221
Telex 339105

Hewlett Packard Ltd
Wedge House
799, London Road
Thornton Heath
Surrey, CR4 6XL
GB England
Tel (01) 684 0103/8
Telex 946825

Hewlett Packard Ltd
14 Wesley St
Castleford
Yorks WF10 1AE
Tel (0977) 550016
TWX 5557335

Hewlett Packard Ltd
Tradax House
St Mary s Walk
Maidenhead
Berkshire, SL6 1ST
GB England

Hewlett Packard Ltd
Morley Road
Staplehill
Bristol BS16 4QT
GB England

Hewlett Packard Ltd
South Queensferry
West Lothian EH30 9TG
GB Scotland
Tel (031) 331 1188
Telex 72682

UNITED STATES

ALABAMA

700 Century Park South
Suite 128
Birmingham 35226
Tel (205) 822 6802

P O Box 4207
8290 Whitesburg Dr
Huntsville 35802
Tel (205) 881 4591

ARIZONA

2336 E Magnolia St
Phoenix 85034
Tel (602) 273 8000

2424 East Aragon Rd
Tucson 85706
Tel (602) 273 8000

'ARKANSAS

Medical Service Only
P O Box 5646
Brady Station
Little Rock 72215
Tel (501) 376 1844

CALIFORNIA

1579 W Shaw Ave
Fresno 93771
Tel (209) 224 0582

1430 East Orangethorpe Ave
Fullerton 92631
Tel (714) 870 1000

5400 West Rosecrans Blvd
P O Box 92105
World Way Postal Center
Los Angeles 90009
Tel (213) 970 7500
TWX 910 325 6608

3939 Lankershim Boulevard
North Hollywood 91604
Tel (213) 877 1282
TWX 910 499 2671

3200 Hillview Av
Palo Alto, CA 94304
Tel (408) 988 7000

646 W North Market Blvd
Sacramento 95834
Tel (916) 929-7222

9606 Aero Drive
P O Box 23333
San Diego 92123
Tel (714) 279 3200

363 Brookhollow Dr
Santa Ana, CA 92705
Tel (714) 641 0977

3003 Scott Boulevard
Santa Clara 95050
Tel (408) 988 7000
TWX 910 338 0518

454 Carlton Court
So. San Francisco 94080
Tel (415) 877 0772

'**Tarzana**
Tel (213) 705 3344

COLORADO

5600 DTC Parkway
Englewood 80110
Tel (303) 771 3455

CONNECTICUT

47 Barnes Industrial Road
Barnes Park South
Wallingford 06492
Tel (203) 265 7801

FLORIDA

P O Box 24210
2727 N W 62nd Street
Ft Lauderdale 33309
Tel (305) 973 2600

4080 Woodcock Drive #132
Brownett Building
Jacksonville 32207
Tel (904) 398 0663

P O Box 13910
6177 Lake Ellenor Dr
Orlando 32809
Tel (305) 859 2900

P O Box 12826
Suite 5 Bldg 1
Office Park North
Pensacola 32575
Tel (904) 476 8422

110 South Hoover Blvd
Suite 120
Tampa 33609
Tel (813) 872 0900

GEORGIA

P O Box 105005
450 Interstate North Parkway
Atlanta 30348
Tel (404) 955 1500
TWX 810 766 4890

Medical Service Only
'**Augusta** 30903
Tel (404) 736 0592

P O Box 2103
1172 N Davis Drive
Warner Robins 31098
Tel (912) 922 0449

HAWAII

2875 So King Street
Honolulu 96826
Tel (808) 955 4455

ILLINOIS

211 Prospect Rd
Bloomington 61701
Tel (309) 663 0383

5201 Tollview Dr
Rolling Meadows
60008
Tel (312) 255 9800
TWX 910 687 2260

INDIANA

7301 North Shadeland Ave
Indianapolis 46250
Tel (317) 842 1000
TWX 810 260 1797

IOWA

2415 Heinz Road
Iowa City 52240
Tel (319) 351 1020

KENTUCKY

10170 Linn Station Road
Suite 525
Louisville 40223
Tel (502) 426 0100

LOUISIANA

P O Box 1449
3229 39 Williams Boulevard
Kenner 70062
Tel (504) 443 6201

MARYLAND

7121 Standard Drive
Parkway Industrial Center
Hanover 21076
Tel (301) 796 7700
TWX 710-862-1943

2 Choke Cherry Road
Rockville 20850
Tel (301) 948 6370
TWX 710 828 9684

MASSACHUSETTS

32 Hartwell Ave
Lexington 02173
Tel (617) 861 8960
TWX 710 326 6904

MICHIGAN

23855 Research Drive
Farmington Hills 48024
Tel (313) 476 6400

724 West Centre Ave
Kalamazoo 49002
Tel (616) 323 8362

MINNESOTA

2400 N Prior Ave
St. Paul 55113
Tel (612) 636 0700

MISSISSIPPI

322 N Mart Plaza
Jackson 39206
Tel (601) 982 9363

MISSOURI

11131 Colorado Ave
Kansas City 64137
Tel (816) 763 8000
TWX 910 771 2087

1024 Executive Parkway
St. Louis 63141
Tel (314) 878 0200

NEBRASKA

Medical Only
7101 Mercy Road
Suite 101
Omaha 68106
Tel (402) 392 0948

NEVADA

'**Las Vegas**
Tel (702) 736 6610

NEW JERSEY

Crystal Brook Professional
Building
Route 35
Eatontown 07724
Tel (201) 542 1384

W 120 Century Rd
Paramus 07652
Tel (201) 265 5000
TWX 710 990 4951

NEW MEXICO

P O Box 11634
Station E
11300 Lomas Blvd N E
Albuquerque 87123
Tel (505) 292 1330
TWX 910 989 1185

156 Wyatt Drive
Las Cruces 88001
Tel (505) 526 2484
TWX 910 9983 0550

NEW YORK

6 Automation Lane
Computer Park
Albany 12205
Tel (518) 458 1550
TWX 710 444 4961

650 Perinton Hill Office Park
Fairport 14450
Tel (716) 223 9950
TWX 510 253 0092

No 1 Pennsylvania Plaza
55th Floor
34th Street & 8th Avenue
New York 10001
Tel (212) 971 0800

5858 East Molloy Road
Syracuse 13211
Tel (315) 455 2486

1 Crossways Park West
Woodbury 11797
Tel (516) 921 0300
TWX 510 221 2183
Tel (513) 671 7400

NORTH CAROLINA

5605 Roanne Way
Greensboro 27409
Tel (919) 852 1800

OHIO

Medical/Computer Only
9920 Carver Road
Cincinnati 45242
Tel (513) 891 9870
[illegible]
Cleveland 44[illegible]
Tel (216) 243 7300
TWX 810 423 9430

962 Crupper Ave
Columbus 43229
Tel (614) 436 1041

330 Progress Rd
Dayton 45449
Tel (513) 859 8202

OKLAHOMA

P O Box 32008
6301 N Meridan Avenue
Oklahoma City 73112
Tel (405) 721-0200

9920 E 42nd Street
Suite 121
Tulsa 74145
Tel (918) 665 3300

OREGON

17890 S W Lower Boones
Ferry Road
Tualatin 97062
Tel (503) 620 3350

PENNSYLVANIA

1021 8th Avenue
King of Prussia Industrial Park
King of Prussia 19406
Tel (215) 265 7000
TWX 510-660 2670

111 Zeta Drive
Pittsburgh 15238
Tel (412) 782-0400

SOUTH CAROLINA

P O Box 6442
6941 0 N Trenholm Road
Columbia 29206
Tel (803) 782 6493

TENNESSEE

8906 Kingston Pike
Knoxville 37919
Tel (615) 691 2371

3070 Directors Row
Directors Square
Memphis 38131
Tel (901) 346 8370

'**Nashville**
Medical Service Only
Tel (615) 244 5448

TEXAS

4171 North Mesa
Suite C110
El Paso 79902
Tel (915) 533 3555

P O Box 42816
10535 Harwin St
Houston 77036
Tel (713) 776 6400

'**Lubbock**
Medical Service Only
Tel (806) 799 4472

P O Box 1270
201 E Arapaho Rd
Richardson 75081
Tel (214) 231 6101

205 Billy Mitchell Road
San Antonio 78226
Tel (512) 434 8241

UTAH

2160 South 3270 West Street
Salt Lake City 84119
Tel (801) 972 4711

VIRGINIA

P O Box 9669
2914 Hungary Spring Road
Richmond 23228
Tel (804) 285 3431

Computer Systems/Medical
Only
Airport Executive Center
Suite 302
5700 Thurston Avenue
Virginia Beach 23455
Tel (804) 460 2471

WASHINGTON

Bellefield Office Pk
1203 114th Ave S E
Bellevue 98004
Tel (206) 454 3971
TWX 910 443 2446

P O Box 4010
Spokane 99202
Tel (509) 535 0864

WEST VIRGINIA

Medical/Analytical Only
4604 MacCorkle Ave S E
Charleston 25304
Tel (304) 925 0492

WISCONSIN

150 South Sunny Slope Road
Brookfield 53005
Tel (414) 784 8800

FOR U.S. AREAS NOT LISTED

Contact the regional office nearest you **Atlanta** Georgia **North Holly-wood**, California **Rockville** Maryland **Rolling Meadows** Illinois Their complete addresses are listed above

USSR

Hewlett Packard
Representative Office
USSR
Pokrovsky Boulevard
4/17 kw 12
Moscow 101000
Tel 294 20 24
Telex 7825 hewpak su

YUGOSLAVIA

Iskra Commerce n sol o
Zastopstvo Hewlett Packard
Obilicev Venac 26
YU 11000 **Beograd**
Tel 636 955
Telex 11530

Iskra Commerce n sol o
Zastopstvo Hewlett Packard
Miklosiceva 38/VII
YU 61000 **Ljubljana**
Tel 321 674, 315-879
Telex 31583

URUGUAY

Pablo Ferrando S A C e I
Avenida Italia 2877
Casilla de Correo 370
Montevideo
Tel 40 3102
Telex 702 Public Booth
Para Pablo Ferrando

VENEZUELA

Hewlett Packard de Venezuela
C A
P O Box 50933
Caracas 105
Los Ruices Norte
3a Transversal
Edificio Segre
Caracas 107
Tel 239 4133 (20 lines)
Telex 25146 HEWPACK

ZAMBIA

R J Tilbury (Zambia) Ltd
P O Box 2792
Lusaka
Tel 73793

MEDITERRANEAN AND MIDDLE EAST COUNTRIES NOT SHOWN, PLEASE CONTACT

Hewlett Packard S A
Mediterranean and Middle East
Operations
35 Kolokotroni Street
Platia Kefallariou
GR Kifissia **Athens**, Greece
Tel 8080359 429
Telex 21 6588
Cable HEWPACKSA Athens

SOCIALIST COUNTRIES NOT SHOWN PLEASE CONTACT

Hewlett Packard Ges m b H
Handelskai 52
P O Box 7
A 1205 **Vienna**, Austria
Tel (0222) 35 16 21 to 27
Cable HEWPAK Vienna
Telex 75923 hewpak a

OTHER AREAS NOT LISTED, CONTACT

Hewlett Packard
Intercontinental
3495 Deer Creek Road
Palo Alto, California 94304
Tel (415) 856 1501
TWX 910 373 1267
Cable HEWPACK Palo Alto
Telex 034 8300, 034 8493

Hewlett Packard S A
7 rue du Bois du Lan
P O Box
CH 1217 Meyrin 2 **Geneva**
[illegible]
Tel (022) 82 70 00
Cable HEWPACKSA Geneva
Telex 2 24 86

Service Only
2 15 80

HEWLETT
PACKARD

1000 N.E. Circle Blvd., Corvallis, OR 97330

82937-90007 Rev. B 10/80

Printed in U.S.A.

FN END

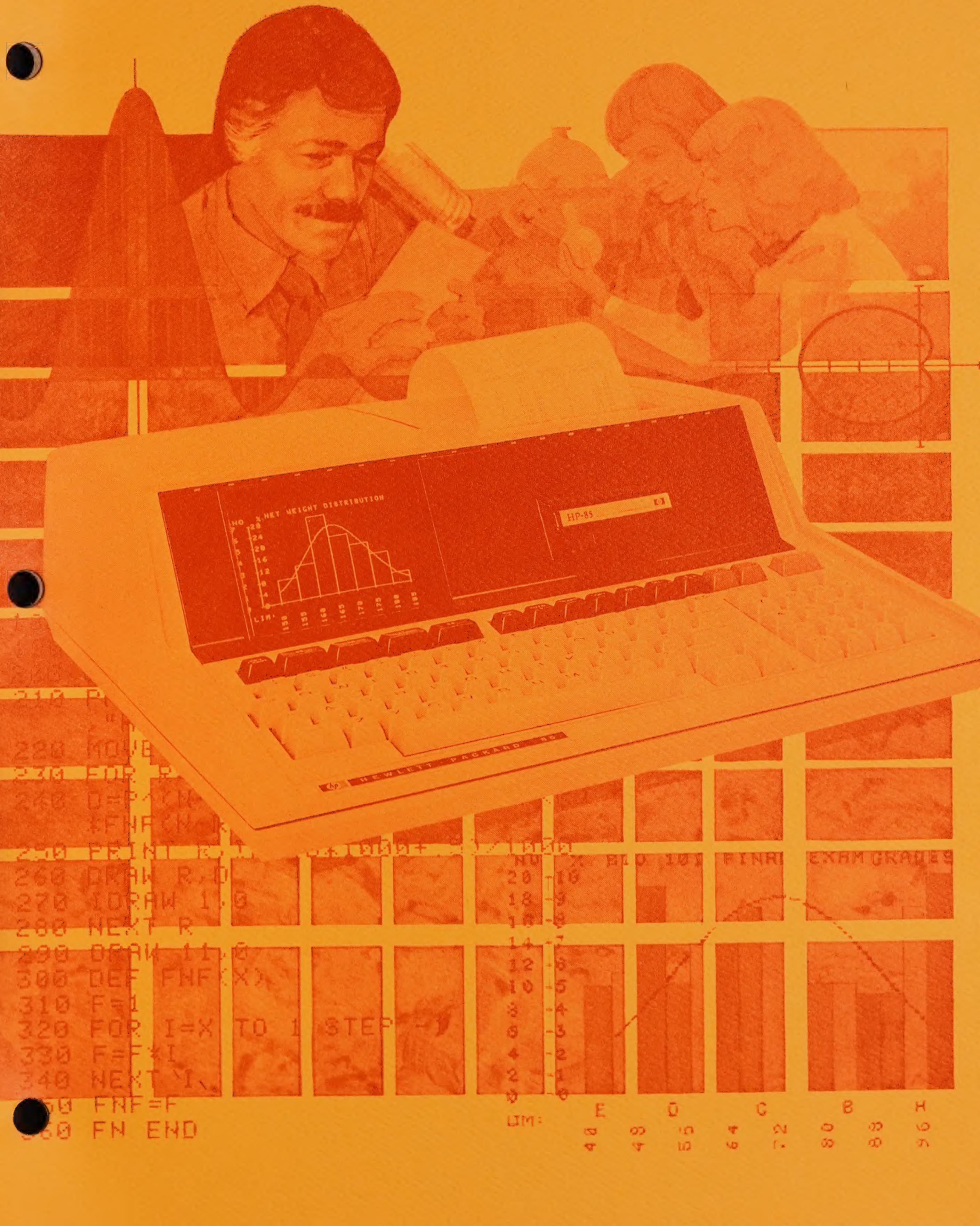
HEWLETT · PACKARD · 85

FN END

FN END

HEWLETT · PACKARD
FN END

NEXT
STEP
FN END


% NET WEIGHT DISTRIBUTION
NO
LIM:
150
155
160
165
170
175
180
185
HP-85
HEWLETT PACKARD 85
hp
220 MOVE
230 FOR R
240 D=P^(N
*FNF(N-K
250 PRINT R,I
260 DRAW R,D
270 IDRAW 1,0
280 NEXT R
290 DRAW 11,0
300 DEF FNF(X)
310 F=1
320 FOR I=X TO 1 STEP -1
330 F=F*I
340 NEXT I
350 FNF=F
360 FN END
NO % BIO 101 FINAL EXAM GRADES
20 10
18 9
16 8
14 7
12 6
10 5
8 4
6 3
4 2
2 1
0 0
E
D
C
B
A
LIM:
40
48
56
64
72
80
88
96